PROJECT REPORT ON

PREDICTION OF CORONARY HEART DISEASE

POST GRADUATE PROGRAM IN DATA SCIENCE ENGINEERING

LOCATION: BANGALORE

BATCH: PGPDSE-FT BANGALORE JAN22

SUBMITTED BY

Group No. 1 [Batch: Jan2022]

GROUP MEMBERS

1. SIMARAN VOHRA
2. JEYAKESAN PARAMASAMY
3. BIDITA SAHA
4. RIYA TALUKDER
5. SUKHESH HEGDE

RESEARCH SUPERVISOR

Mr. Srikar Muppidi

GREAT LAKES EXECUTIVE LEARNING

Great Lakes Institute of Management

ABSTRACT

The field of medical analysis is often referred to as a valuable source of rich information. Yet,Heart Disease Diagnosing is one of the most challenging and difficult diagnostic tests reported by NCBI (National Center for Biotechnology Information) and it has been a very big challenge for the Medical community and Health Care Sector to Accurately predict and help the patients with Coronary Heart Diseases.

According to the Centers for Disease Control and Prevention (CDC), about half the population of America has at least one of the risk factors for heart disease, Considering all the data collected by them as on February 2022 and analyzing the key factors It is possible to create an opportunity for diagnosing the early onset of heart disease.

Coronary Heart Disease (CHD) is one of the major causes of death all around the world therefore early detection of CHD can help reduce these rates. The challenge lies in the complexity of the data and correlations when it comes to prediction using conventional techniques.

The aim of this research is to use the historical medical data to predict CHD using Machine Learning (ML) technology. Logistic Regression and Supervised Machine Learning Models have been used to identify the best learner and achieve an accuracy of more than 95% in classifying the binary output data correctly, wherein our challenge is to predict if a person has CHD or not!

The code has been written in python language, the most widely used open source programming language used in the field of Data Analytics and Data Science.

We implemented this project with the following,

Techniques:

- Predictive Modelling
- Supervised Machine Learning Classification Models
- Tuning
- Boosting

- Tools:
  - Python
  - IDE - Jupyter Notebook

- Domain:
  - Data Analytics - Healthcare Sector.

ACKNOWLEDGEMENT

We hereby certify that the work done by us for the implementation and completion of this project is absolutely original and to the best of our knowledge. It is a team effort and each of the members has equally contributed to the project. Specially, we would like to thank our mentor(Mr. Srikar Muppidi) for helping us with this project.He allowed us to work on this project.At last, We would like to extend our heartfelt thanks to our teachers because without their help this project would not have been successful.

Date: 15.07.2022

Place: Bangalore.

CERTIFICATE OF COMPLETION

This is to certify that the project titled “Prediction of Coronary Heart Disease” for case resolution was undertaken and completed under the supervision of Mr. Srikar Muppidi for Post Graduate Program in Data Science and Engineering (PGP – DSE)

Mentor: Mr. Srikar Muppidi

TABLE OF CONTENTS

ABBREVIATIONS

S. No.	Full Form	Abbreviation
1.	National Center for Biotechnology Information	NCBI
2.	Coronary heart disease	CHD
3.	Myocardial infarction	MI
4.	Behavioral Risk Factor Surveillance System	BRFSS
5.	Exploratory Data Analysis	EDA
6.	Centers for Disease Control and Prevention	CDC
7.	K - Nearest Neighbours	KNN
8.	Probability values	P-VALUES

CHAPTER 1:INTRODUCTION

EXECUTIVE SUMMARY

Heart Disease Diagnosing is one of the most difficult Diagnostic tests reported by NCBI (National Center for Biotechnology Information) and it has been a very big challenge for the Medical community and Health Care Sector to Accurately predict and help the patients with Coronary Heart Diseases.

The most recent dataset (as of February 15, 2022) includes data from 2020. It consists of 401,958 rows and 279 columns formed by 274 continuous numerical , 1 discrete numerical and 5 categorical variables.

_MICHD is our target variable;that provides data on the survey question “If the respondents have ever reported having coronary heart disease (CHD) or myocardial infarction (MI)?”

From our calculation we got that 10% of the respondents have been affected by CHD and the remaining 90% of people were not affected. Based on the presence of corresponding _MICHD data record for all the other variables records were retained and the rows where the _MICHD values was missing/blank all those were dropped or deleted, _MICHD had about 3571 records Missing and hence all those corresponding rows under all the other independent variables were deleted.Thus leaving the dataset with about 398387 records and 41 attributes.

PROBLEM STATEMENT

To identify the key indicators that have been causing or contributing to Coronary Heart Disease and then create a predictive model to predict whether a person will be affected by Coronary Heart Disease or not? Considering those key indicators as input data from the patient or the user.

DATA AND FINDINGS

The dataset was collected from the CDC which is also a major part of the Behavioral Risk Factor Surveillance System (BRFSS), conducts annual telephone surveys to gather data on the health status of U.S. residents. As the CDC describes: "Established in 1984 with 15 states, BRFSS now collects data in all 50 states as well as the District of Columbia and three U.S. territories. BRFSS completes more than 400,000 adult interviews each year, making it the largest continuously conducted health survey system in the world.".

The most recent dataset (as of February 15, 2022) includes data from 2020. It consists of 401,958 rows and 280 columns.

The vast majority of columns are questions asked to respondents about their health status, such as "Do you have serious difficulty walking or climbing stairs?" or "Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes]".

From this dataset, It was noticed that many different factors (questions) that directly or indirectly influence heart disease were present, which led to the consideration of selecting the questions by relevance to the problem statement so that it would help the models to learn to produce accurate results.

Dataset Source:

https://drive.google.com/file/d/1rRAeREz417lGWXjnqzJ7JQFrkqsvsaPx/view?usp=sharing

https://www.cdc.gov/brfss/annual_data/annual_2020.html

Dataset Dimension:

- **Shape of the Dataset :** (401958, 280)

  Total Records : 401958
  Total Attributes : 280

- **Target Variable :** _MICHD

The dataset was in a different file format with columns in abbreviated form, through the CDC websites' manual on dataset description (Code Book) the target variable and the understanding of other variables was established.

Target Variable

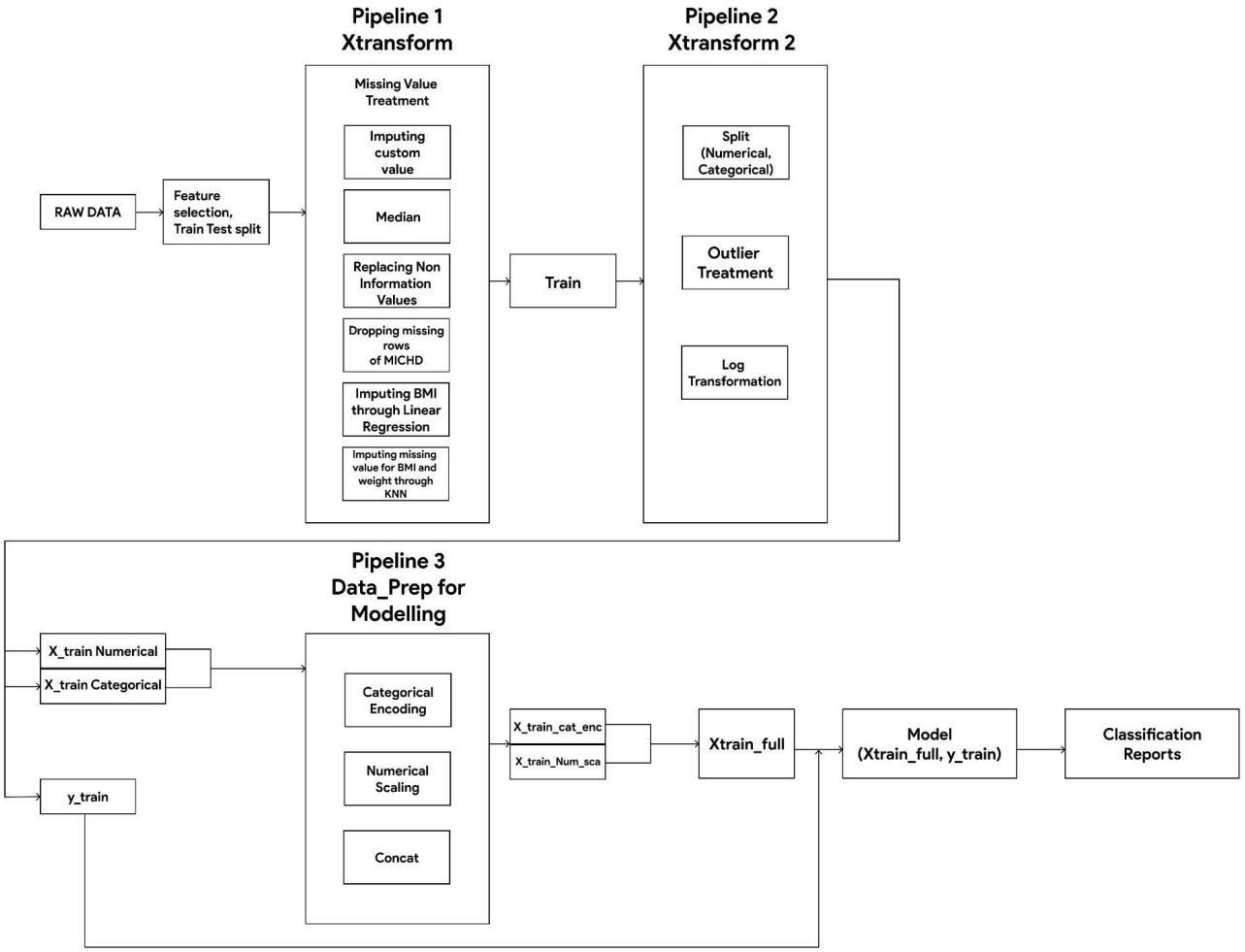
The target variable in our dataset is _MICHD, that provides that data on the survey question "If the respondents have ever reported having coronary heart disease (CHD) or myocardial infarction (MI)?". As our problem statement is to create a machine learning model to predict if a person would be diagnosed with CHD or not, we have taken _MICHD as the target variable with data values 0 and 1.
where,

* 0 - Indicates that the respondents never reported Heart Disease
* 1- Indicates that the respondents had Heart Disease.

CHAPTER 2: GENERAL APPROACH,DATA PROCESSING AND FEATURE SELECTION

Approach-Methodology

The Approach that we chose to work was an iterative process, where we define the core problem, accordingly to which we Collect Raw-Data from relevant sources followed up by Data Pre-Processing and actual Data Processing through which we'll improve the quality of data by handling the missing values , treating the outliers and further do a feature selection process if required and then build models with different implementations and finally evaluating the results for each model to finalise the best model with most accurate results and prediction based on the Performance reports of each model.



Data Pre-Processing

As discussed earlier, since the dataset has the most number of attributes , we had to check for the relevance of the variables with regard to the problem statement and thereby based on the relevance and irrelevance certain features were retained and most of them were dropped.

In [4]:
```
# Part 1 - dropping columns based on relevance

cols_drop = ['Unnamed: 0', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE', 'SEQNO', '_PSU', 'QSTLANG', 'MSCODE', '_STST
             '_WT2RAKE', '_CHISPNC', '_CRACE1', '_CPRACE', '_CLLCPWT', '_DUALUSE', '_DUALCOR', '_LLCPWT2', '_LLCPWT', '_ALTETH3',
             '_RFMAM22', '_MAM5023', '_RFPAP35', '_RFPSA23', '_CLNSCPY', '_SGMSCPY', '_SGMS10Y', '_RFBLDS4', '_STOLDNA', '_VIRCOL
             'CTELENM1', 'PVTRESD1', 'COLGHOUS', 'STATERE1', 'CELPHONE', 'LADULT1', 'COLGSEX', 'NUMADULT', 'LANDSEX', 'NUMMEN',
             'SAFETIME', 'CTELNUM1', 'CELLFON5', 'CADULT1', 'CELLSEX', 'PVTRESD3', 'CCLGHOUS', 'CSTATE1', 'LANDLINE', 'POORHLTH',
             'DIABAGE3', 'RMVTETH4', 'RENTHOM1', 'NUMHHOL3', 'NUMPHON3', 'CPDEMO1B', 'STOPSMK2', 'ALCDAY5', 'FLSHTMY3', 'FALL12MN
             'HOWLONG', 'HADPAP2', 'LASTPAP2', 'HPVTEST', 'HPLSTTST', 'HADHYST2', 'PCPSAAD3', 'PCPSADI1', 'PCPSARE1', 'PSATEST1',
             'COLNSCPY', 'COLNTEST', 'SIGMSCPY', 'SIGMTEST', 'BLDSTOL1', 'LSTBLDS4', 'STOOLDNA', 'SDNATEST', 'VIRCOLON', 'VCLNTES
             'PREDIAB1', 'INSULIN1', 'BLDSUGAR', 'FEETCHK3', 'DOCTDIAB', 'CHKHEMO3', 'FEETCHK', 'EYEEXAM1', 'DIABEYE', 'DIABEDU',
             'WORKCFS', 'TOLDHEPC', 'TRETHEPC', 'PRIRHEPC', 'HAVEHEPC', 'HAVEHEPB', 'MEDSHEPB', 'HLTHCVR1', 'CIMEMLOS', 'CDHOUSE'
             'CDSOCIAL', 'CDDISCUS', 'CAREGIV1', 'CRGVREL4', 'CRGVLNG1', 'CRGVHRS1', 'CRGVPRB3', 'CRGVALZD', 'CRGVPER1', 'CRGVHOU
             'LCSLAST', 'LCSCTSCN', 'CNCRAGE', 'CNCRTYP1', 'CSRVTRT3', 'CSRVDOC1', 'CSRVSUM', 'CSRVRTRN', 'CSRVINST', 'CSRVINSR',
             'CSRVPAIN', 'CSRVCTL2', 'PCPSADE1', 'PCDMDEC1', 'HPVADVC4', 'HPVADSHT', 'TETANUS1', 'IMFVPLA1', 'BIRTHSEX', 'SOMALE'
             'ACEDEPRS', 'ACEDRINK', 'ACEDRUGS', 'ACEPRISN', 'ACEDIVRC', 'ACEPUNCH', 'ACEHURT1', 'ACESWEAR', 'ACETOUCH', 'ACETTHE
             'RCSRLTN2', 'CASTHDX2', 'CASTHNO2']
```

Dropping those columns we were left with 106 features , Again on analysing we found that many variables had data spread out , that is data was spread under various other categories of the same particular subject hence the columns were feature engineered and were dropped which belonged to those engineered variables leaving us with 41 variables.

In [7]:
```
# Part 2 - dropping feature engineered variables

cols_drop_engineered_vars = ['_URBSTAT', 'ASTHMA3', 'ASTHNOW', '_LTASTH1', '_CASTHM1', 'HAVARTH4', '_DENVST3', 'LASTDEN4', '_PRAC
                             '_HISPANC', '_RACE', '_RACEG21', '_RACEGR3', '_RACEPRV', '_AGEG5YR', '_AGE65YR', '_AGE_G', 'HTM4',
                             '_RFSMOK3', 'SMOKE100', 'SMOKDAY2', 'LASTSMK2', 'USENOW3', 'ECIGARET', 'ECIGNOW', 'LCSNUMCG', 'MAXDR
                             '_DRNKDRV', '_RFSEAT2', '_RFSEAT3', 'SEATBELT', 'DRNKDRI2', 'SEXVAR', 'DECIDE', '_MENT14D', '_RFHLTH
                             '_HCVU651', 'HLTHPLN1', 'PERSDOC2', 'MEDCOST', 'CHECKUP1', 'EXERANY2', 'CNCRDIFF', 'DIFFDRES', 'DIFF
                             'SHINGLE2', 'PNEUVAC4', '_AIDTST4', 'HIVTST7', 'HIVRISK5', 'USEMRJN2', 'RSNMRJN1', 'EDUCA', 'VETERAN
                             'INCOME2', '_CHLDCNT', 'PREGNANT', 'HEIGHT3', 'WEIGHT2']

print('Number of columns to be dropped which belong to engineered variables: ', len(cols_drop_engineered_vars))
```

```
Number of columns to be dropped which belong to engineered variables:  65
```

In [8]:
```
data.drop(cols_drop_engineered_vars, axis=1, inplace=True)

data.shape
```

Out[8]: (401958, 41)

Train-Test Split of the Data:

The whole dataset is separated into the Train and Test Dataset at the initial stage itself so as to avoid the model from over fitting and also at the same time providing the model the scope to learn and predict with proper accuracy.

Splitting the train and test data

```
In [44]: train, test = train_test_split(data, test_size=0.3, random_state=42)

In [45]: print(train.shape)
         print(test.shape)

         (281370, 41)
         (120588, 41)
```

Null and Missing Values Treatment:

In our approach handling the missing values has been established as a separate pipeline , i.e Pipeline 1 (X-Transform()) , Through the X-Transform() function we will be treating the missing values for both the Train and the Test Dataset , Initially the treatment is started with train dataset and the pipeline includes the following approaches.

The total Missing values present in the above retained 41 columns of the Train are as follows:

Attributes	No. of Missing Values
_STATE	0
HHADULT	124753
GENHLTH	8
PHYSHLTH	5
MENTHLTH	5
SLEPTIM1	3
CVDSTRK3	3
CHCSCNCR	3

CHCOCNCR	3
CHCCOPD2	5
ADDEPEV3	6
CHCKDNY2	6
DIABETE4	6
MARITAL	12
EMPLOY1	2925
WEIGHT2	9852
DEAF	12415
BLIND	13324
DIFFWALK	15280
AVEDRNK3	211099
DRNK3GE5	211626
MARIJAN1	271339
_METSTAT	7127
_IMPRACE	0
_TOTINDA	0
_MICHD	3571
_ASTHMS1	0
_DRDXAR2	2303
_EXTETH3	0
_SEX	0
_AGE80	0
HTIN4	23808
WTKG3	34948
_BMI5	41357
_EDUCAG	0
_INCOMG	0
_SMOKER3	0
DRNKANY5	0
DROCDY3_	0
_DRNKWK1	0
_RFDRHV7	0

Of the 41 columns , we got around 27 columns that has missing values

Replacing the missing data with custom values :

The attributes EMPLOY1, AVEDRNK3, DRNK3GE5 and MARIJAN1 holding missing values were replaced by the custom value ‘0’ as it holds a significant meaning corresponding to their data values.

Replacing the missing values with Median:

These were the variables with missing values treated by replacing them with the Median value so as to maintain the distribution of data.

'HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH', 'SLEPTIM1', 'CVDSTRK3', 'CHCSCNCR', 'CHCOCNCR', 'CHCCOPD2', 'ADDEPEV3', 'CHCKDNY2', 'DIABETE4', 'MARITAL', 'WEIGHT2', 'DEAF', 'BLIND', 'DIFFWALK', '_METSTAT', '_DRDXAR2', 'HTIN4'

```
def replace_with_custom(data):
    # replace missing data with custom values
    data['EMPLOY1'].fillna(0, inplace=True)
    data['AVEDRNK3'].fillna(0, inplace=True)
    data['DRNK3GE5'].fillna(0, inplace=True)
    data['MARIJAN1'].fillna(0, inplace=True)

    return data

# function to replace missing values with median or mode

def replacena_with_median_mode(data):

    replacena_with_median_cols = ['HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH', 'SLEPTIM1', 'HTIN4']

    replacena_with_mode_cols = ['CVDSTRK3', 'CHCSCNCR', 'CHCOCNCR', 'CHCCOPD2', 'ADDEPEV3', 'CHCKDNY2', 'DIABETE4',
                                'MARITAL', 'DEAF', 'BLIND', 'DIFFWALK', '_METSTAT', '_DRDXAR2']

    for col in data.columns:

        if col in replacena_with_median_cols:
            data[col].fillna(data[col].median(), inplace=True)
        elif col in replacena_with_mode_cols:
            data[col].fillna(data[col].mode()[0], inplace=True)

    return data
```

Replacing the no-information values with the custom values:

Almost all the variables along with the actual Missing values (Nan) , Have certain values that do not form an actual interpreting data in regard to the dataset perspective.

Such as the categories : Don't Know, Not sure , Refused to answer etc.,

All these are also considered to be Missing values and they have been replaced with the custom values that correspond to the other actual values present within the data under that particular category.

```
# replace non-information values with custom values based on codebook
def replace_no_info_values(data, train):
    data = data.replace({'HHADULT': {77: 1, 99: 1},
                         'GENHLTH': {7: train['GENHLTH'].mode()[0], 9: train['GENHLTH'].mode()[0]},
                         'PHYSHLTH': {77: 0, 88: 0, 99: train['PHYSHLTH'].median()},
                         'MENTHLTH': {77: 0, 88: 0, 99: train['MENTHLTH'].median()},
                         'SLEPTIM1': {77: train['SLEPTIM1'].median(), 99: train['SLEPTIM1'].median()},
                         'CVDSTRK3': {7: 2, 9: 2},
                         'CHCSCNCR': {7: 2, 9: 2},
                         'CHCOCNCR': {7: 2, 9: 2},
                         'CHCCOPD2': {7: 2, 9: 2},
                         'ADDEPEV3': {7: 2, 9: 2},
                         'CHCKDNY2': {7: 2, 9: 2},
                         'DIABETE4': {3: 2, 4: 2, 7: 2, 9: 2},
                         'MARITAL': {9: train['MARITAL'].mode()[0]},
                         'EMPLOY1': {9: 0},
                         'DEAF': {7: 2, 9: 2},
                         'BLIND': {7: 2, 9: 2},
                         'DIFFWALK': {7: 2, 9: 2},
                         'AVEDRNK3': {88:0, 77: 0, 99: 0},
                         'DRNK3GE5': {88:0, 77: 0, 99: 0},
                         'MARIJAN1': {77: 0, 88:0, 99: 0},
                         '_TOTINDA': {9: train['_TOTINDA'].mode()[0]},
                         '_ASTHMS1': {9: train['_ASTHMS1'].mode()[0]},
                         '_EXTETH3': {9: train['_EXTETH3'].mode()[0]},
                         '_SEX': {2: 0}, # replacing female (2) with 0
                         '_EDUCAG': {9: train['_EDUCAG'].mode()[0]},
                         '_INCOMG': {9: train['_INCOMG'].mode()[0]},
                         '_SMOKER3': {9: train['_SMOKER3'].mode()[0]},
                         'DRNKANY5': {7: 2, 9: 2},
                         'DROCDY3_': {900: 0},
                         'DRNKWK1': {99900: 0},
                         '_RFDRHV7': {9: train['_RFDRHV7'].mode()[0]},
                         '_RFBING5': {9: train['_RFBING5'].mode()[0]}
                         })
    return data
```

Each variable imputed with which data value is explained and given in the Feature Description Link.

Feature Description Link :

https://docs.google.com/spreadsheets/d/1RS4rXNoqxktYSshWlmflPMXhmqFvZjOtQAz1kChlrTY/edit#gid=112559524

Dropping the Missing Rows:

The Missing value records present in the target variable (_MICHD) are dropped along with their corresponding records belonging to other attributes.

Thus after executing the row drop of the target variable we are left with about 398387 records and 41 attributes.

```
# dropping all rows where _MICHD is missing

def drop_rows_michdna(data):
    drop_ind = data.index[data['_MICHD'].isna()]
    data.drop(drop_ind, axis=0, inplace=True)

    return data
```

After the missing value treatment:

```
_AGE80          0
HTIN4           0
WTKG3       23954
_BMI5       28435
_EDUCAG         0
_INCOMG         0
_SMOKER3        0
DRNKANY5        0
DROCDY3_        0
_RFBING5        0
_DRNKWK1        0
_RFDRHV7        0
```

Except for the variables WTKG3(weight) and BMI5 (BMI) all the other variables were treated and to handle BMI and the Weight the following method was used.

Imputation of BMI and WTKG3 Missing values :

The BMI Variable present in the dataset has about approximately 28,500 records of missing values and in reference to the EDA (Exploratory Data Analysis) It is evidently known that the variable (WTKG3) weight that has about approximately 24,000 missing value records is in high correlation with the BMI.

Thereby, the idea of Imputing the BMI values through the LinearRegression() model with the aid of the corresponding records of the (WTKG3) weight variable, contributed in filling and replacing the 4,500 missing records of the BMI variable approximately.

Following that it was necessary to also fill the remaining 24,000 missing values present in both the BMI and WTKG3 variables , and this was imputed through the KNN() Imputer which is nothing but the K-Nearest Neighbour Algorithm.

Implementation of LinearRegression()

In [82]:
```
# function to impute all nan values of _BMI5, where WTKG3 is not na

def train_linear_model(data):

    # Selecting all non-nan pairs of _BMI5 and WTKG3
    data_impute_wtkg3_mv = data[['_BMI5', 'WTKG3']][~data[['_BMI5', 'WTKG3']].isna().any(axis=1)]
    print('Shape of the overall data with non-nan pairs of _BMI5 and WTKG3', data_impute_wtkg3_mv.shape)

    # splitting the above data into train and test
    X_train_mv, X_test_mv, y_train_mv, y_test_mv = train_test_split(data_impute_wtkg3_mv['WTKG3'].values.reshape(-1, 1),
                                                                    data_impute_wtkg3_mv['_BMI5'],
                                                                    test_size=0.3, random_state=42)
    print('Shape of the training set: ', X_train_mv.shape)
    print('Labels of training set: ', X_test_mv.shape)
    print('Shape of the test set: ', y_train_mv.shape)
    print('Labels of the test set: ', y_test_mv.shape)

    linear_reg_mv = LinearRegression()

    linear_reg_mv.fit(X_train_mv, y_train_mv)

    print('Score of the Linear model is: ', linear_reg_mv.score(X_test_mv, y_test_mv))

    return linear_reg_mv

# imputing all missing BMI5 values with the predicted value using WTKG3
def impute_bmi5_lin_reg(model, data):
    X_final_mv = data[data['_BMI5'].isna() & ~data['WTKG3'].isna()]['WTKG3']
    print(X_final_mv.shape)


    for row in X_final_mv.index:
        test_data_wtkg3 = X_final_mv.loc[row]
        predicted_bmi = model.predict(test_data_wtkg3.reshape(-1, 1))
        data.loc[row]['_BMI5'] = predicted_bmi

    return data
```

In [83]:
```
# storing the linear model
linear_reg_mv = train_linear_model(train)

train = impute_bmi5_lin_reg(linear_reg_mv, train)
```

Implementation of KNN():

```
In [52]: from sklearn.impute import KNNImputer

def train_imputeKNN(data):
    missing_data = data.copy()

    knn_imp = KNNImputer()
    data = knn_imp.fit(missing_data)

    return knn_imp

def imputeKNN(model, data):
    data_trans = knn_imp.transform(data)
    return data_trans
```

```
In [53]: knn_imp = train_imputeKNN(train)

train_trans = imputeKNN(knn_imp, train)
```

Outlier Treatment and Transformations:

The outlier treatment and the corresponding variable transformations are connected under a specific user defined function which is held by the pipeline 2 , X-Transform2()and the outliers were treated using the capping strategy as in our dataset we had values behaving in a way that after a certain value, the lower bound outliers and the higher bound outliers exhibited the same behaviour.

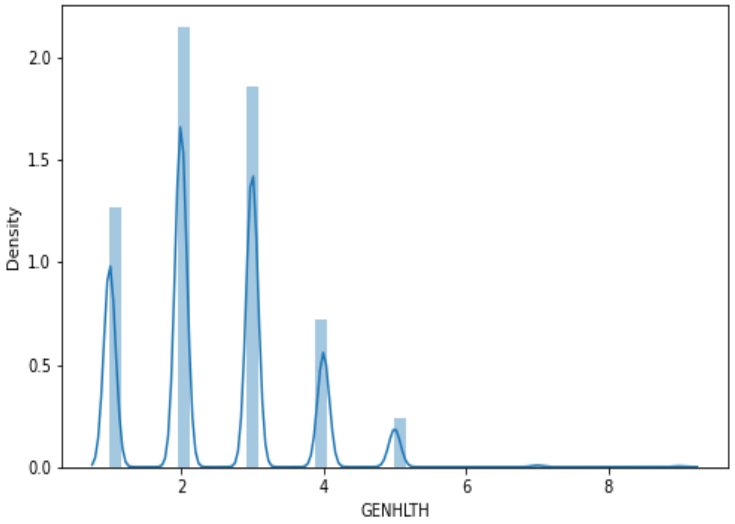
Outliers of a few features such as physical health , mental health , and drinking pattern were replaced manually.

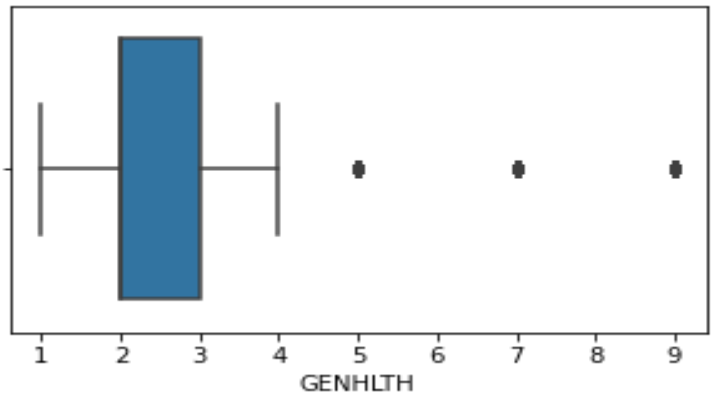
And corresponding numerical variables were log transformed in order to handle the skewness present in the variables.

Outlier Visualization:

The visualization is provided for few of the variables.

Variable Name: **GENHLTH**

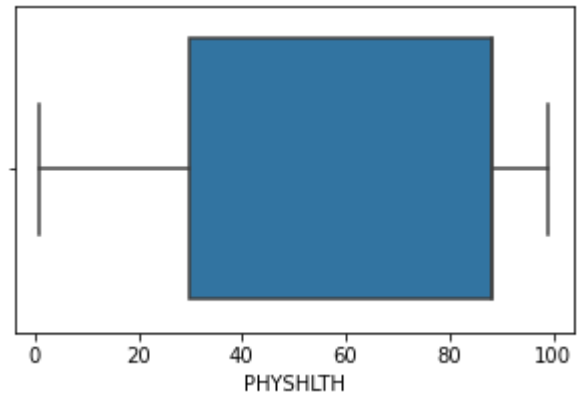




Skewness value of the variable GENHLTH is 0.639474451249044
Kurtosis value of the variable GENHLTH is 0.8135901743270457
From the distribution plot and the skewness value we could see that the variable GENHLTH Is having more data distributed in the left region and hence its Right skewed

The variable GENHLTH is Platy Kurtic
The tail indicates the presence of slight larger outliers

Variable Name: **PHYSHLTH**



Skewness value of the variable PHYSHLTH is -1.0340143626144718

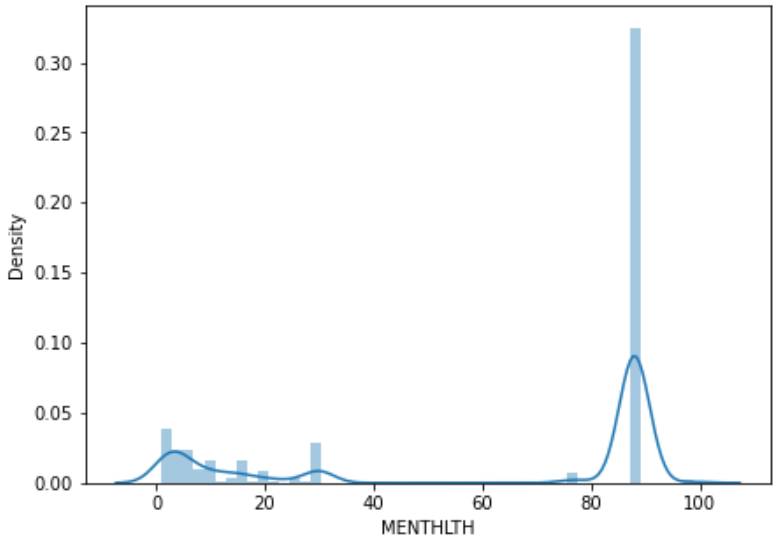
Kurtosis value of the variable PHYSHLTH is -0.8172374994355596

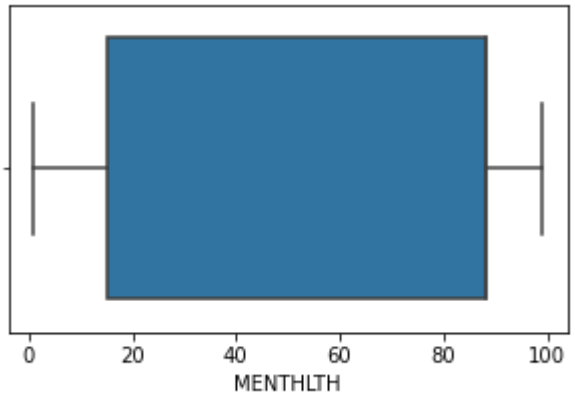
From the distribution plot and the skewness value we could see that the variable PHYSHLTH Is having more data distributed in the right region and hence its Left skewed

The variable PHYSHLTH is Lepto Kurtic

The flat tails indicates the presence of small outliers

Variable Name: **MENTHLTH**





Skewness value of the variable MENTHLTH is -0.7258386835057923

Kurtosis value of the variable MENTHLTH is -1.3825364418460586

From the distribution plot and the skewness value we could see that the variable MENTHLTH Is having more data distributed in the right region and hence its Left skewed

The variable MENTHLTH is Lepto Kurtic

The flat tails indicates the presence of small outliers

Final List of Features

By deploying various methods and approaches of EDA Techniques we were able to understand the structure of the data and the nature of the variables. Analysis like Univariate and Bi-variate were done , also checking out the correlations , presence of muliti-collinearity was also done through various visualisation techniques to finalise the following 41 features in the dataset.

'_STATE', 'HHADULT', 'GENHLTH', 'PHYSHLTH', 'MENTHLTH', 'SLEPTIM1', 'CVDSTRK3', 'CHCSCNCR', 'CHCOCNCR', 'CHCCOPD2', 'ADDEPEV3', 'CHCKDNY2', 'DIABETE4', 'MARITAL', 'EMPLOY1', 'DEAF', 'BLIND', 'DIFFWALK', 'AVEDRNK3', 'DRNK3GE5', 'MARIJAN1', '_METSTAT', '_IMPRACE', '_TOTINDA', '_MICHD', '_ASTHMS1', '_DRDXAR2', '_EXTETH3', '_SEX', '_AGE80', 'HTIN4', 'WTKG3', '_BMI5', '_EDUCAG', '_INCOMG', '_SMOKER3', 'DRNKANY5', 'DROCDY3_', '_RFBING5', '_DRNKWK1', '_RFDRHV7'

CHAPTER 3 : STATISTICAL ANALYSIS

Statistical analysis :

All the features were put into statistical tests to find the proportionality of each independent variable with the target variable which gives an idea if the features were actually making an impact on our target variable.

Two Sample Proportion Test:

The feature's significance was also established through these statistical tests(two sample proportion tests) by the consideration of the probability values (P-values). That is, features with p-value below 0.05 were selected to be significant features.

Algorithm implementation:

```
[ ]  def check_proportionality_michd(col, class_names):

         class_names = class_names
         df = pd.crosstab(data[col], data['_MICHD'])

         proportion_michd = df[1.0].values / df.sum(axis=1).values

         proportion_michd = pd.Series(data=proportion_michd, index=class_names).sort_values()

         sns.barplot(x=proportion_michd.index , y=proportion_michd.values)
         plt.rcParams['figure.figsize'] = [8, 5]
         plt.xticks(rotation=90)
         plt.show()

         z_prop, p_val = two_sample_prop_test(df)

         if p_val < 0.05:
             print('The Alternate Hypothesis passed the 2 sample proportion test.')
             print('The two classes in this category are shown to have different proportions of _MICHD')
         else:
             print('The Null Hypothesis failed to be rejected.')
             print('The two classes in this category cannot be shown to have different proportions of _MICHD')

         return proportion_michd

     def two_sample_prop_test(df):

         # Getting rid of unknown values
         if 9.0 in df.index:
             df = df.drop(9.0, axis=0)

         if len(df.index) > 2:
             return 0, 0

         # Calculating count and nobs for the category
         count = df[1.0].values
         nobs = df.sum(axis=1)

         z_prop, p_val = proportions_ztest(count=count, nobs=nobs, alternative='two-sided')

         return z_prop, p_val
```

Visualization of Correlation between the numerical Features:

	HHADULT	GENHLTH	PHYSHLTH	MENTHLTH	SLEPTIM1	WEIGHT2	AVEDRNK3	DRNK3GE5	MARIJAN1	_AGE80	HTIN4	WTKG3	_BMI5	_INCOMG	DROCDY3_	_DRNKWK1
HHADULT	1	-0.03 9351	-0.02 559	0.009 972	-0.01 2738	0.007 515	0.013 787	0.005 209	0.002 64	-0.17 0277	0.009 43	0.004 993	0.003 35	0.081 529	-0.02 5205	-0.00 1932
GENHLTH	-0.03 9351	1	0.469 408	0.233 143	-0.06 426	0.147 152	-0.06 4682	-0.00 2256	0.031 771	0.191 287	-0.07 3972	0.135 856	0.203 315	-0.25 3252	-0.11 439	-0.03 4206
PHYSHLTH	-0.02 559	0.469 408	1	0.283 506	-0.06 0904	0.064 915	-0.04 5116	0.000 682	0.036 718	0.109 067	-0.04 7292	0.053 805	0.090 454	-0.18 0285	-0.06 5746	-0.01 878
MENTHLTH	0.009 972	0.233 143	0.283 506	1	-0.11 6486	0.023 142	0.039 575	0.065 074	0.077 24	-0.14 8061	-0.05 7294	0.013 363	0.048 034	-0.13 9537	0.003 399	0.036 98
SLEPTIM1	-0.01 2738	-0.06 426	-0.06 0904	-0.11 6486	1	-0.04 7021	-0.02 652	-0.02 5328	-0.02 2015	0.096 565	-0.00 9	-0.04 5936	-0.04 5972	0.013 104	0.011 057	-0.01 1297
WEIGHT2	0.007 515	0.147 152	0.064 915	0.023 142	-0.04 7021	1	0.062 114	0.036 676	-0.00 0873	-0.05 4549	0.466 525	0.973 988	0.837 274	0.009 742	-0.01 655	0.029 731
AVEDRNK3	0.013 787	-0.06 4682	-0.04 5116	0.039 575	-0.02 652	0.062 114	1	0.463 818	0.063 291	-0.15 7475	0.136 291	0.066 818	-0.00 2155	0.039 898	0.403 765	0.771 048
DRNK3GE5	0.005 209	-0.00 2256	0.000 682	0.065 074	-0.02 5328	0.036 676	0.463 818	1	0.067 059	-0.10 0839	0.087 412	0.037 6	-0.00 7252	-0.00 4712	0.378 657	0.568 399
MARIJAN1	0.002 64	0.031 771	0.036 718	0.077 24	-0.02 2015	-0.00 0873	0.063 291	0.067 059	1	-0.07 2579	0.028 02	-0.00 2172	-0.01 8801	-0.04 8529	0.041 885	0.056 255
_AGE80	-0.17 0277	0.191 287	0.109 067	-0.14 8061	0.096 565	-0.05 4549	-0.15 7475	-0.10 0839	-0.07 2579	1	-0.12 2872	-0.04 965	0.009 646	-0.02 789	0.009 304	-0.05 3658
HTIN4	0.009 43	-0.07 3972	-0.04 7292	-0.05 7294	-0.00 9	0.466 525	0.136 291	0.087 412	0.028 02	-0.12 2872	1	0.494 614	0.002 451	0.120 4	0.129 772	0.109 969
WTKG3	0.004 993	0.135 856	0.053 805	0.013 363	-0.04 5936	0.973 988	0.066 818	0.037 6	-0.00 2172	-0.04 965	0.494 614	1	0.847 214	0.019 226	-0.00 9806	0.032 994
_BMI5	0.003 35	0.203 315	0.090 454	0.048 034	-0.04 5972	0.837 274	-0.00 2155	-0.00 7252	-0.01 8801	0.009 646	0.002 451	0.847 214	1	-0.05 7298	-0.08 8424	-0.02 6933
_INCOMG	0.081 529	-0.25 3252	-0.18 0285	-0.13 9537	0.013 104	0.009 742	0.039 898	-0.00 4712	-0.04 8529	-0.02 789	0.120 4	0.019 226	-0.05 7298	1	0.118 921	0.029 967
DROCDY3_	-0.02 5205	-0.11 439	-0.06 5746	0.003 399	0.011 057	-0.01 655	0.403 765	0.378 657	0.041 885	0.009 304	0.129 772	-0.00 9806	-0.08 8424	0.118 921	1	0.565 848
_DRNKWK1	-0.00 1932	-0.03 4206	-0.01 878	0.036 98	-0.01 1297	0.029 731	0.771 048	0.568 399	0.056 255	-0.05 3658	0.109 969	0.032 994	-0.02 6933	0.029 967	0.565 848	1

Chi-Square Test:

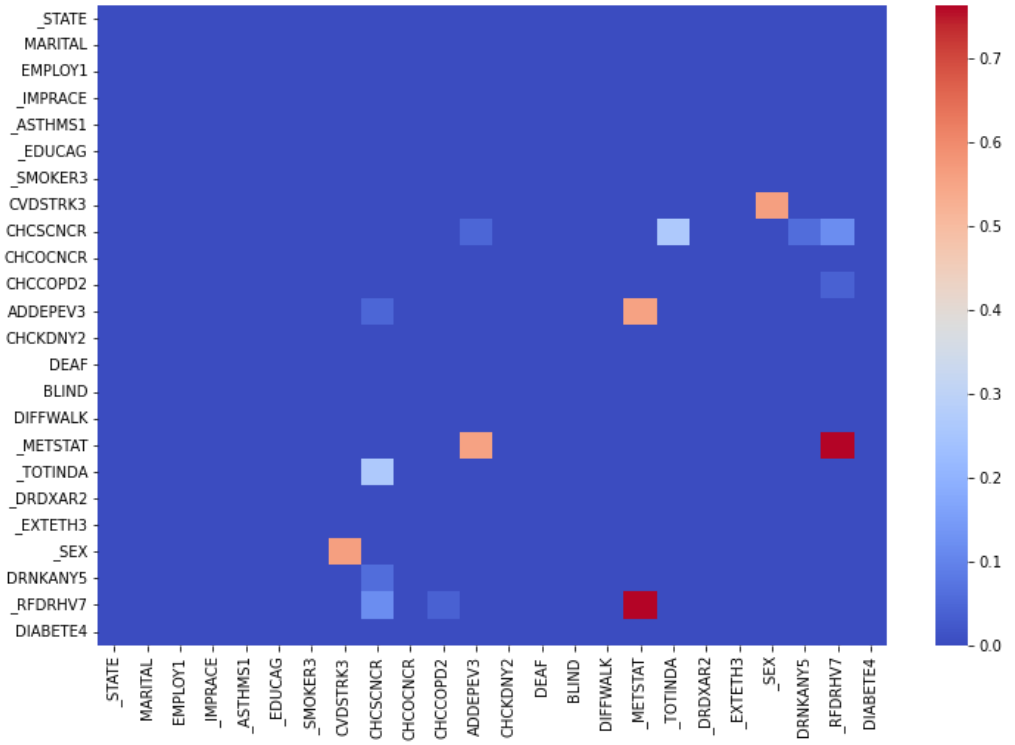
Chi2 tests were also done to determine if a difference between observed data and expected data is due to chance, or if it is due to a relationship between the variables and accordingly the features were selected and treated.

Algorithm Implementation:

```
[ ]  from scipy.stats import chi2_contingency
```

```
[ ]  row_pval = []

     df_chi2 = pd.DataFrame()

     for col_a in X_cat.columns:
         row_pval = []

         for col_b in X_cat.columns:

             ct_table_ind = pd.crosstab(X_cat[col_a], X_cat[col_b]).values
             chi2_stat, p, dof, expected = chi2_contingency(ct_table_ind)

             row_pval.append(p)

         row_pval = pd.Series(row_pval)
         df_chi2 = pd.concat([df_chi2, row_pval], axis=1)

     df_chi2
```

CHi2 Test - Visualization:



CHAPTER 4:MODEL BUILDING AND METHODS

Model Building and Methods

Splitting Data:

It is very essential to prepare the data all set and ready for the model to adapt, hence the numerical variables have to be scaled and the categorical variables should be encoded combining which forms the fully trained X.

Before performing the actions above, the dataset should be split into Numerical Variables set and categorical variables set.

Feature Encoding and Scaling of the Data:

Encoding and Scaling was established as a separate pipeline that is the 3rd Pipeline where all the categorical variables are dummy encoded and all the numerical variables being scaled through the StandardScaler algorithm, thereby finally combining both the encoded categorical variables and the scaled numerical variables we obtain the fully cleaned and equipped X_train.

Hence moving ahead passing the X_train_full and the y_train obtained from the pipeline 2 through the models we'll be able to arrive at the best model for our problem statement.

Preparing the data for model building

```
# pipeline to encode categoric, scale numeric and join both dataframes

scale = StandardScaler()
scale.fit(X_numeric)

def data_prep_modelling(X_numeric, X_categoric):
    global scale

    X_categoric_encoded = pd.get_dummies(X_categoric.astype('object'), drop_first=True)
    print('Shape of the encoded categoric data: ', X_categoric_encoded.shape)

    X_numeric_scaled = scale.transform(X_numeric)
    print('Shape of the numeric data: ', X_numeric_scaled.shape)

    X_full = pd.concat([X_categoric_encoded, pd.DataFrame(X_numeric_scaled, columns=X_numeric.columns, index=X_numeric.index)], a
    print('Shape of the full transformed data: ', X_full.shape)

    return X_categoric_encoded, X_numeric_scaled, X_full
```

```
# using the data_prep_modelling function to ready the data for modelling
X_train_categoric_encoded, X_train_numeric_scaled, X_train_full = data_prep_modelling(X_train_numeric, X_train_categoric)
```

```
Shape of the encoded categoric data:  (278882, 96)
Shape of the numeric data:  (278882, 15)
Shape of the full transformed data:  (278882, 111)
```

Classification Models and Techniques:

Recursive model building is one of the key highlights of our methodology as we are going to build models, evaluate their performance, find out the features or hyper parameters behind the performance of the model and then fine tune the possible causes and make our model better up to a certain threshold which in real world will be decided based on the actual business requirements.

Multiple classification models including

Random Forest

Decision Trees

Logistic Regression

AdaBoost Classifier, etc.

Are implemented recursively in the process to compare the results of each.

CHAPTER 5 : MODEL EVALUATION AND RESULT

Random Forest :

The Baseline models were built to get the idea of the minimum amount of performance that could be obtained from each of the models we have decided to use in the future.

The Base model that was built in our case was Random Forest which gave an accuracy score of 92% and recall value of about 0.05/1.0 , As our main objective was to predict the CHD of the patient , ' recall ' is our key indicator to maximize, as we dont want to miss a person who likely has the heart disease unpredicted.

Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Random Forest - default	0.047582	0.916171	0.076885	0.955029	0.003717	0.831034	0.51761	0.11

Classification Metrics:

Confusion Matrix

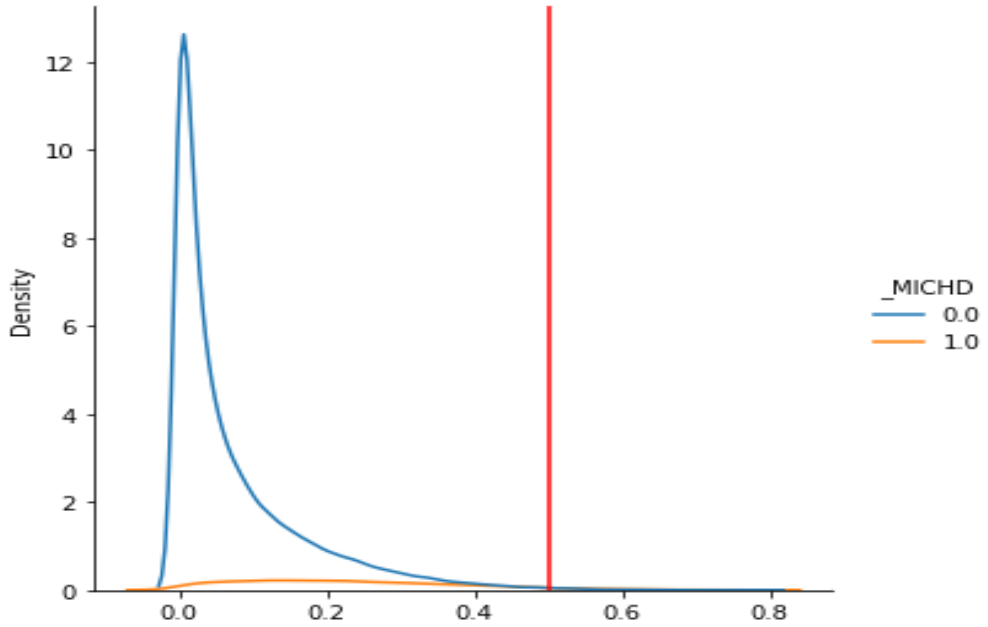
[[109002 310]

[9708 485]]

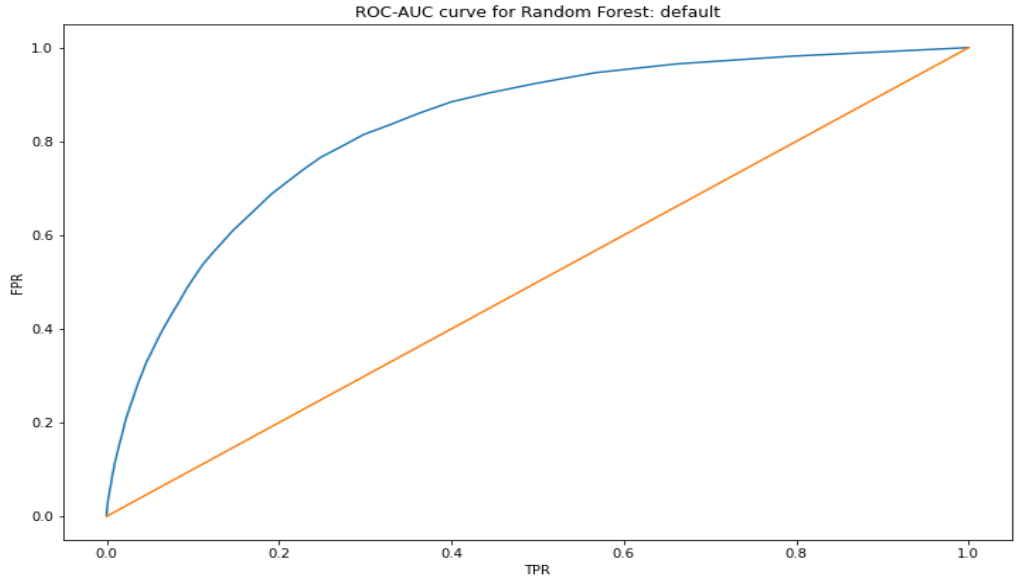
Classification report on Test Dataset

	precision	recall	f1-score	support
0.0	0.92	1.00	0.96	109312
1.0	0.61	0.05	0.09	10193

Distribution of probabilities for the response class

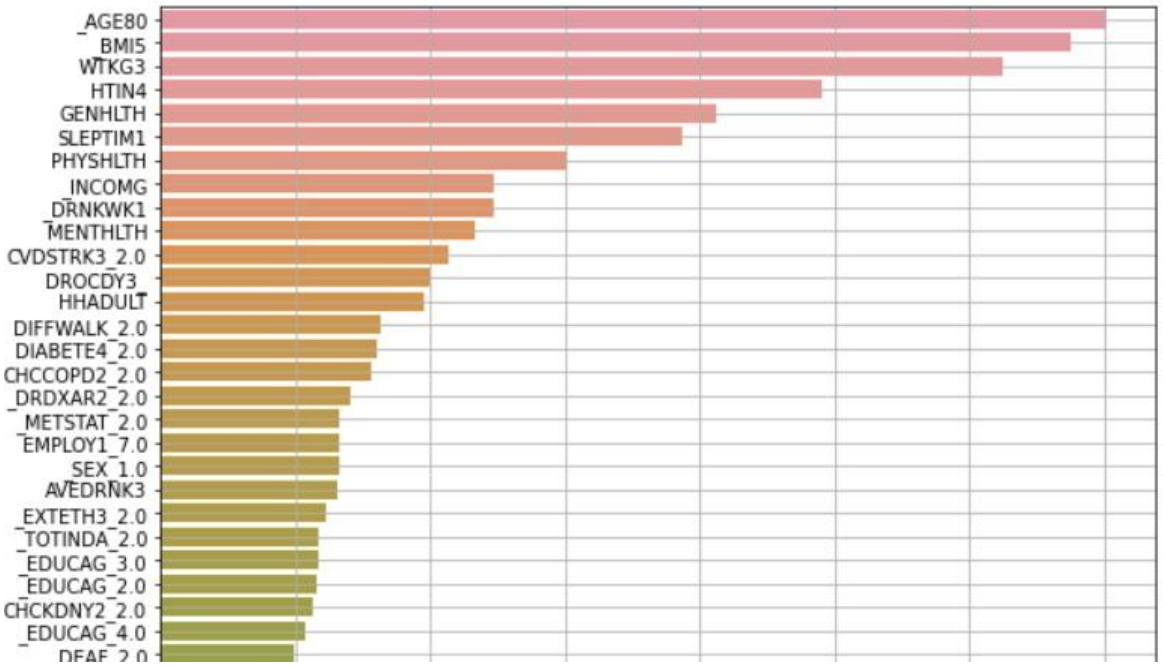


AUC-ROC Curve:



Feature Selection :

According to the default selection within the Random Forest, the most important features are: _AGE80, BMI, WTKG3, _HTIN4 and GENHLTH - which is in accordance with basic domain research found on the topic.



The Significance range for the above visualization ranges from 0.00 to 0.07 with _Age80 (Age) having the maximum significance value 0.07.

Hyperparameter Tuning:

Grid search was used for tuning the random forest where the estimators range given was between 1-30 , with depth range between 0-80 and of which the best estimator suggested by the system was 1.

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
params = {'n_estimators': [1, 2, 5, 10], 'max_depth': [None, 10, 25, 50, 60, 70, 80],
          'max_features': ['auto', 10, 20, 30, 40]}

GS_rf = GridSearchCV(estimator=RandomForestClassifier(),
                     param_grid=params,
                     scoring='recall',
                     cv=kfold,
                     n_jobs=-1,
                     verbose=2)

GS_rf.fit(X_train_full, y_train)
```

Running the tuned random forest with the 1 as its estimator yielded the following result, where the recall value raised nearly 20% compared to our base model with tuning , the recall value 0.23/1.0 , with an accuracy score of 87%.

Confusion Matrix
[[101486 7826]
[7860 2333]]

Classification report

	precision	recall	f1-score	support
0.0	0.93	0.93	0.93	109312
1.0	0.23	0.23	0.23	10193

The model performance report was not only taken considering only the tests but also with the training data it was measured to see the prediction output.

Performance report on the Train data,where we were able to get the desired boosted recall value of about 0.68/1.0 but then it could also be inferred that the Random Forest model is clearly overfitting the training data.

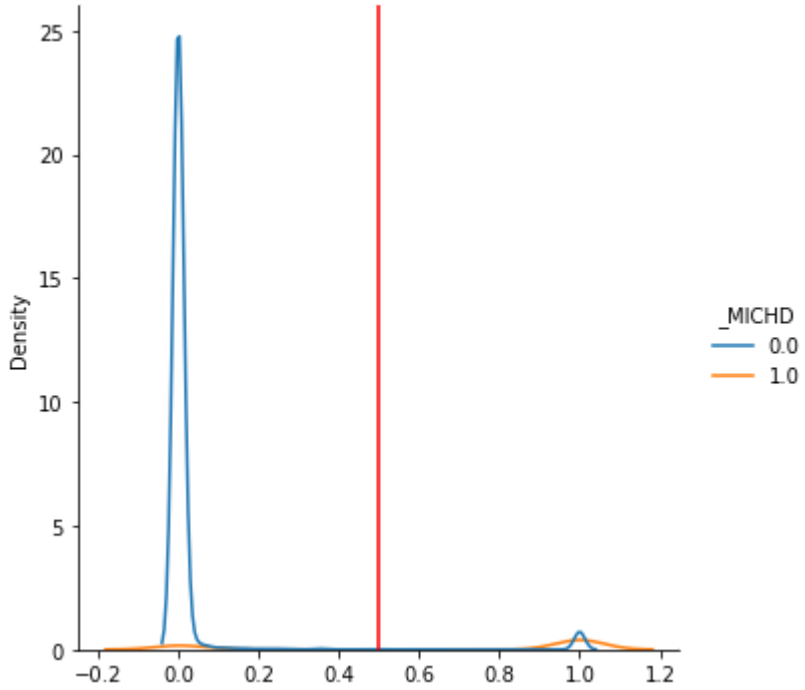
Confusion Matrix
[[248088 6824]
[7599 16371]]

Classification report

	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	254912
1.0	0.71	0.68	0.69	23970

accuracy			0.95	278882
macro avg	0.84	0.83	0.83	278882
weighted avg	0.95	0.95	0.95	278882

Distribution of probabilities for the response class

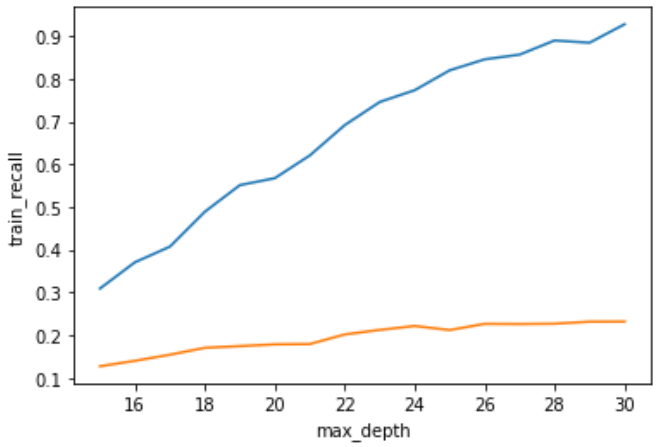


Summary of Random Forest

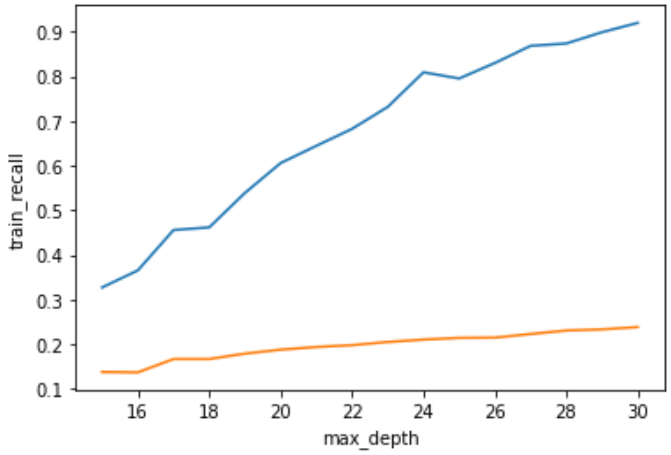
To confirm if the model was overfitting , visualization was made depicting the relationship between the train data score, test data score and the max-depth for each of the feature size values.

The Max_Features range was given : 35-45

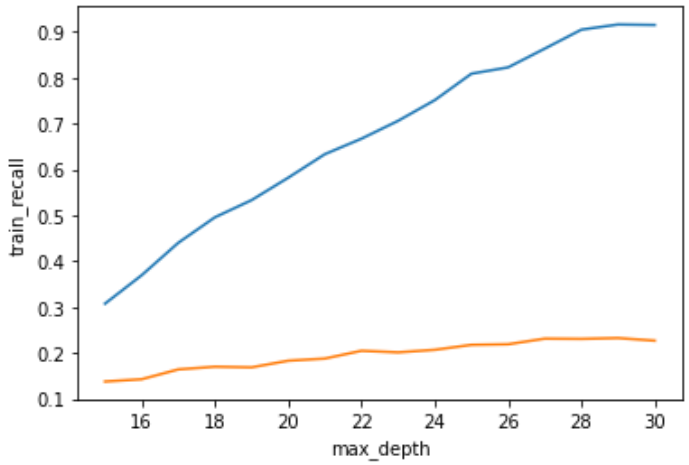
Max_Features : 35



Feature Size : 36



Feature Size :37



It could be observed that , as max_depth increases, the train and test scores also increases, as well the gap between them. (The model increasingly overfits)

Decided on trying a few more runs of the model with higher max depth, as there seems to be an upward trend in that direction, keeping max.features constant.

Hence clearly all the models are overfitting.

Decision Tree

The decision tree model is the model of computation in which the algorithm is considered to be basically a decision tree,that is a sequence of queries or tests that are done adaptively, so the outcome of the previous tests can influence the test performed next.

The relationship between the recall and max depth was visualized

from which we could observe that there is no significant increase to be obtained by increasing the max depth, thus this is the best possible decision tree we can obtain.

Hyper Parameter tuning:

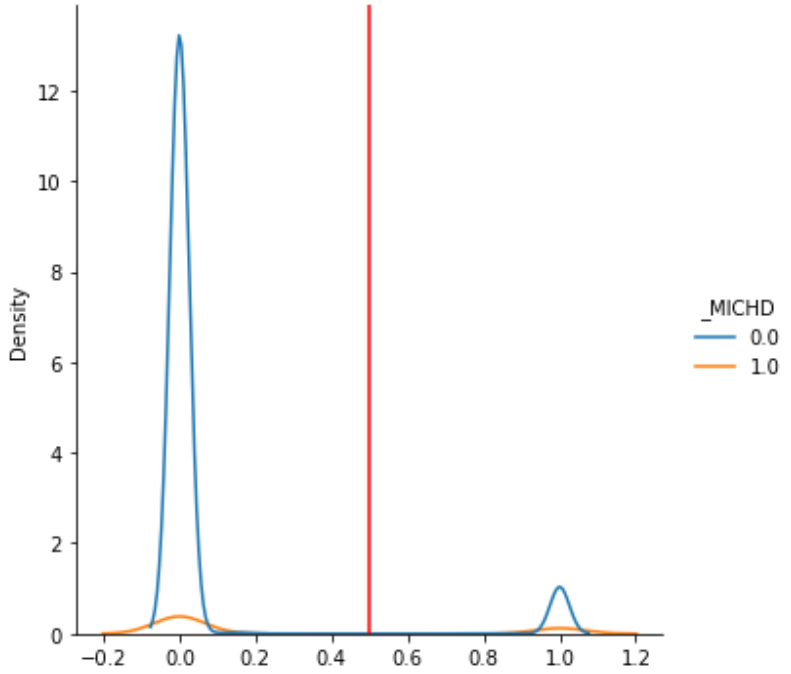
Based on the above results the Decision Tree was tuned,

```
y_preds_dt_tuned = dt_tuned.predict(X_test_full)
y_prob_preds_dt_tuned = dt_tuned.predict_proba(X_test_full)[:, 1]

fpr, tpr, thresh = roc_curve(y_test, y_prob_preds_dt_tuned)

plt.title('ROC-AUC curve for Decision Tree [max_depth=30, max_features=30]')
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1])
plt.xlabel('TPR')
plt.ylabel('FPR')
plt.show()
```

and the model performance reports was generated in which the recall value is 0.24/1.0

Classification report

	precision	recall	f1-score	support
0.0	0.93	0.93	0.93	109312
1.0	0.24	0.24	0.24	10193



score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Decision Tree: max_depth=40, max_features=30	0.239576	0.868976	0.166089	0.762071	0.002981	0.585773	0.171922	0.020833

Decision Tree VS with Random Forest

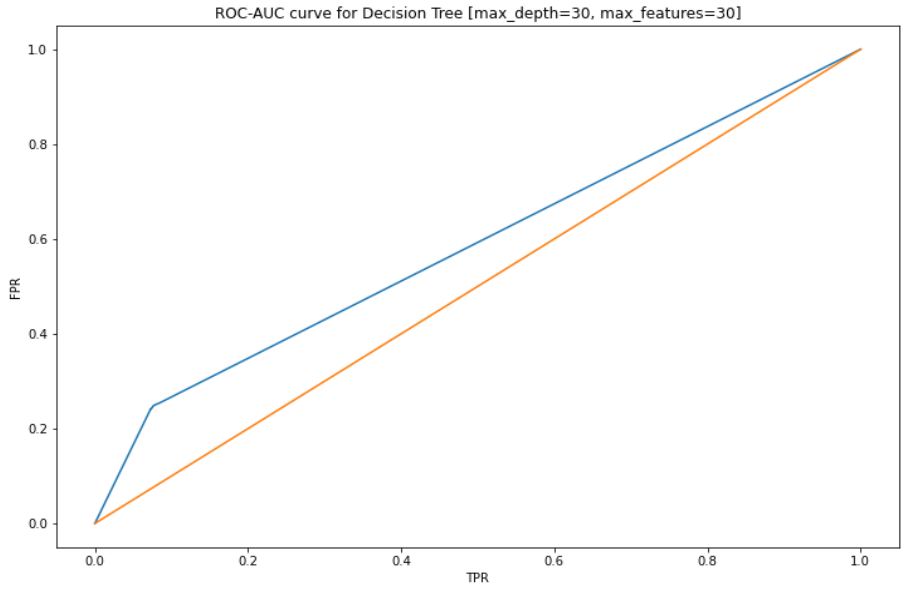
The Tuned Decision tree has a much better performance than the Random Forest in terms of the Recall and Cohen Kappa Scores.

The Decision Tree's bias and variance errors are also much lower than the Random Forest.

The variance error seems to be very low for the Decision tree, but the tree is clearly overfitting on the training data

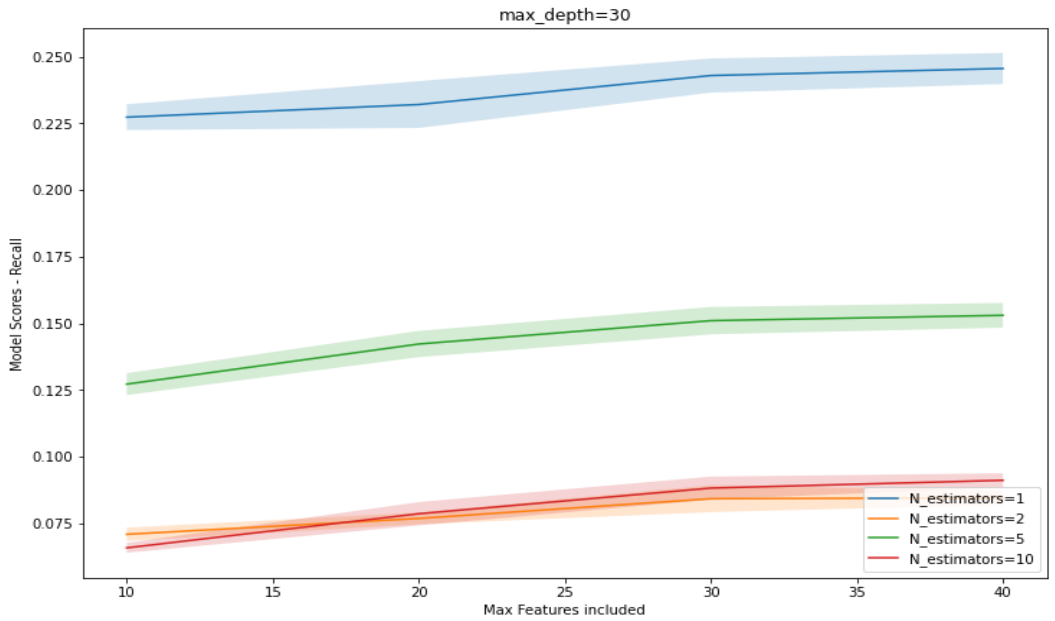
Decreasing the max_depth of the tree decreases the gap between the train and test scores at the cost of the overall decrease in test scores. (Refer graphs above)

AUC-ROC Curve:



On comparing the ROC curves of the tuned Decision Tree and the tuned Random Forest, we observe that the Random Forest has a higher potential if we try to find a better threshold.

According to the scorecard, the best threshold is 0.11



From the above visualisation, we can observe and confirm that the N_estimator=1 is the best fit value among the given estimator ranges, which essentially gives an inference to try the Decision Tree which has the Max-depth value to be 1. Hence we'll proceed further tuning the Decision Tree to check and confirm, as discussed above a decrease in max_depth will aid in decreasing the gap between the train and test scores.

Tuning Other Parameters of Decision Tree:

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

params = {'criterion': ['gini', 'entropy'],
          'max_depth':[None, 15, 25, 50],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 5, 10],
          'max_features' : [None, 10, 20, 30, 40]}

GS_dt = GridSearchCV(estimator=DecisionTreeClassifier(),
                     param_grid=params,
                     scoring='recall',
                     cv=kfold,
                     n_jobs=-1,
                     verbose=2)

GS_dt.fit(X_train_full, y_train)
```

score:

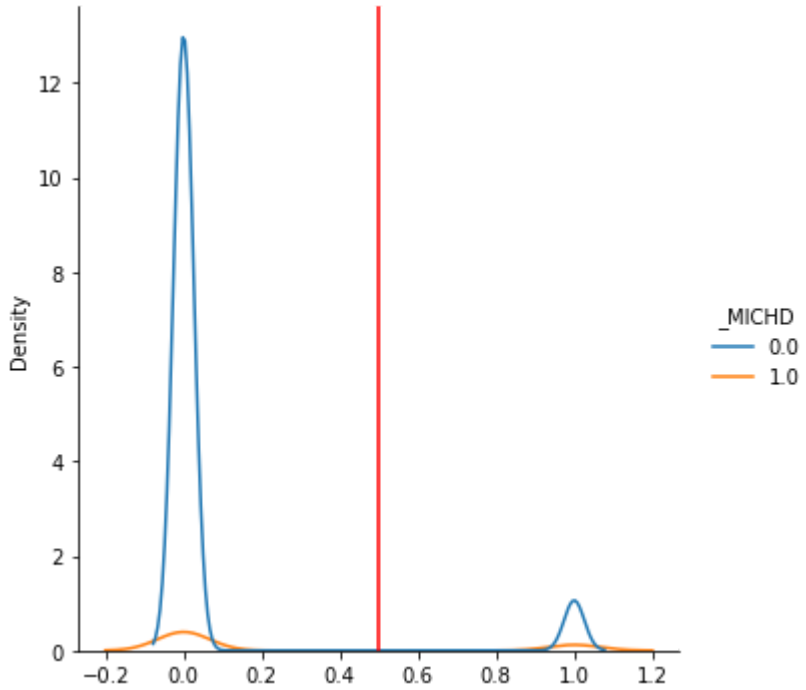
	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Decision Tree - tuned (best=default)	**0.234072**	**0.86545**	**0.156114**	**0.766063**	**0.010158**	**0.579372**	**0.158743**	**1**

Model Performance Report :

Classification report

	precision	recall	f1-score	support
0.0	0.93	0.92	0.93	109268
1.0	0.23	0.23	0.23	10249

Distribution of probabilities for the response class



Logistic Regression

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables.

Assumptions

1. The dependent to predict should be a binary output.

2. Linearity of the independent variables and the log(odds) - most of the variables in our dataset are categoric, this will be a check for the 18 numeric variables existing in the dataset.

3. No strongly influential outliers - we can use the Cook's distance to verify this (or absolute standardized residuals greater than 3)

4. Absence of multicollinearity - will be checked using VIF

5. Independence of observations

6. Sufficiently large sample size - fulfilled

Advantages

1. The Logistic Model is a good choice due to its ease of interpretability.

2. Although the number of features is large, as the sample size is also proportionately large, there is a low risk of the model overfitting the data

Dis-Advantages

1. Constructs linear decision boundaries, which could lead to highly biased models which are not good at identifying complex relationships in the data.

Logistic Regression Using Stats Models

```
Xc_train = sm.add_constant(X_train_full)
logit = sm.Logit(y_train, Xc_train)

logit = logit.fit()

logit.summary()
```

Logit Regression Results

Dep. Variable:	_MICHD	**No. Observations:**	278870
Model:	Logit	**Df Residuals:**	278758
Method:	MLE	**Df Model:**	111
Date:	Fri, 15 Jul 2022	**Pseudo R-squ.:**	0.2283
Time:	09:05:00	**Log-Likelihood:**	-62968.
converged:	True	**LL-Null:**	-81598.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

Logistic Regression - Normal Fit

The basic logistic regression fit was carried out - LogisticRegression()

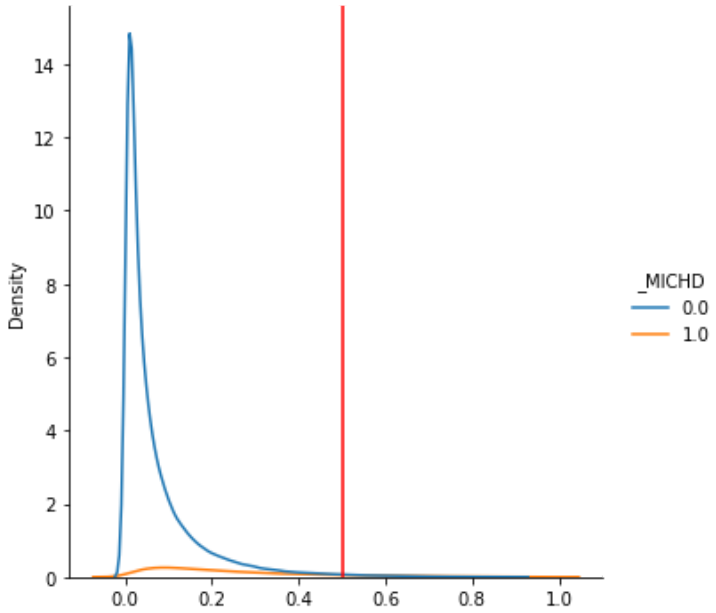
The performance reports were generated for both the Train and Test datasets.

Performance report of the Train-Dataset:

Classification report

	precision	recall	f1-score	support
0.0	0.92	0.99	0.96	254956
1.0	0.55	0.12	0.19	23914

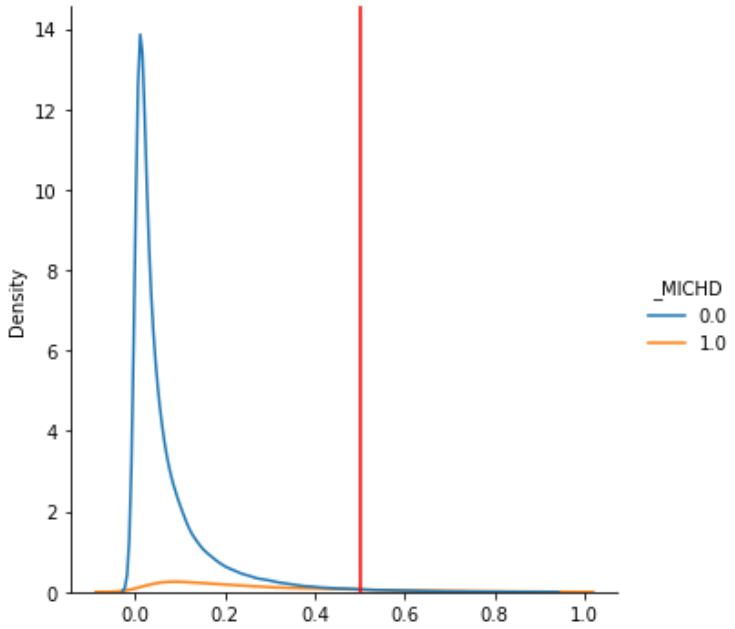
Distribution of probabilities for the response class



Performance report of the Test-Dataset:

	precision	recall	f1-score	support
0.0	0.92	0.99	0.96	109268
1.0	0.56	0.12	0.19	10249

Distribution of probabilities for the response class



Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Logistic Regression - default settings	0.115133	0.916489	0.167015	0.883549	0.004	0.844329	0.536723	0.077961

Inference from the above Logistic Regression - Normal Fit Results:

Considering both the Train and Test results we could observe that the model is not causing overfit over the Train Data and hence there might not be any need of Regularization.

Regularized Logistic Regression Using Statsmodels

```
logit_reg = sm.Logit(y_train.values, Xc_train)

logit_reg = logit_reg.fit_regularized(method='l1')
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.2257972729372056
            Iterations: 314
            Function evaluations: 314
            Gradient evaluations: 314
```

```
logit_reg.summary()
```

Logit Regression Results

Dep. Variable:	y	**No. Observations:**	278870
Model:	Logit	**Df Residuals:**	278758
Method:	MLE	**Df Model:**	111
Date:	Fri, 15 Jul 2022	**Pseudo R-squ.:**	0.2283
Time:	09:05:36	**Log-Likelihood:**	-62968.
converged:	True	**LL-Null:**	-81598.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

Inference:

As expected the regularized model is same as the non-regularized model.

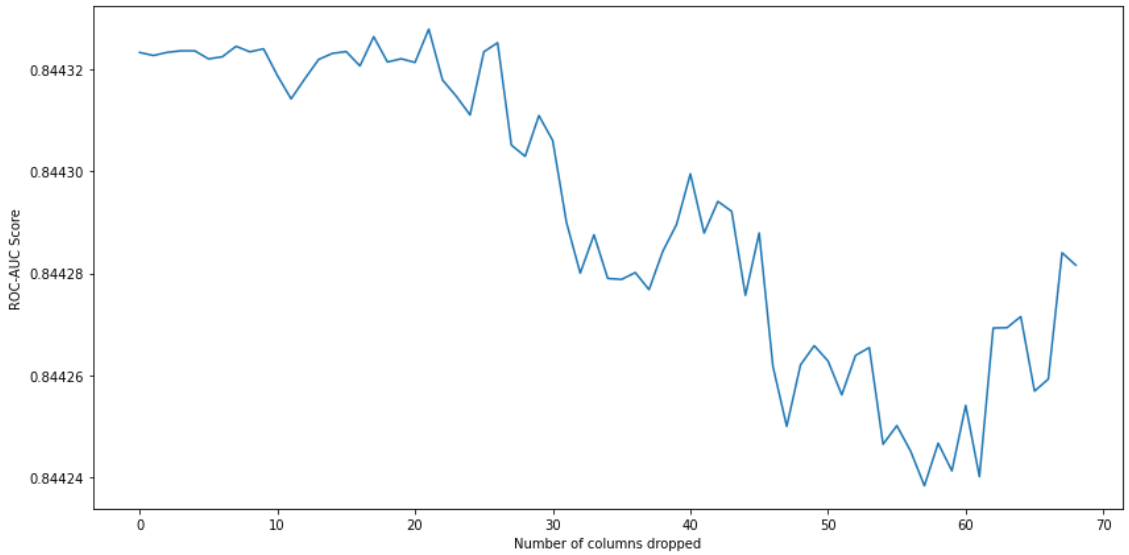
Recursive Feature Selection

To hence our Logistic Regression model for even more better results we chose to do a revised feature selection - Logistic model , where the function eliminates the features with p-values greater than 0.05
The remaining columns were :

['const', '_STATE_4.0', '_STATE_6.0', '_STATE_8.0', '_STATE_15.0', '_STATE_41.0', '_STATE_49.0', '_STATE_53.0', '_STATE_72.0', 'MARITAL_3.0', 'MARITAL_5.0', 'EMPLOY1_1.0', 'EMPLOY1_7.0', 'EMPLOY1_8.0', '_IMPRACE_2.0', '_IMPRACE_3.0', '_IMPRACE_5.0', '_ASTHMS1_3.0', '_EDUCAG_3.0', '_SMOKER3_4.0', 'CVDSTRK3_2.0', 'CHCSCNCR_2.0', 'CHCCOPD2_2.0', 'ADDEPEV3_2.0', 'CHCKDNY2_2.0', 'DEAF_2.0', 'BLIND_2.0', 'DIFFWALK_2.0', '_TOTINDA_2.0', '_DRDXAR2_2.0', '_EXTETH3_2.0', '_SEX_1.0', 'DRNKANY5_2.0', '_RFDRHV7_2.0', 'DIABETE4_2.0', '_RFBING5_2.0', 'HHADULT', 'GENHLTH', 'SLEPTIM1', '_AGE80', 'HTIN4', 'WTKG3', '_INCOMG', '_DRNKWK1']

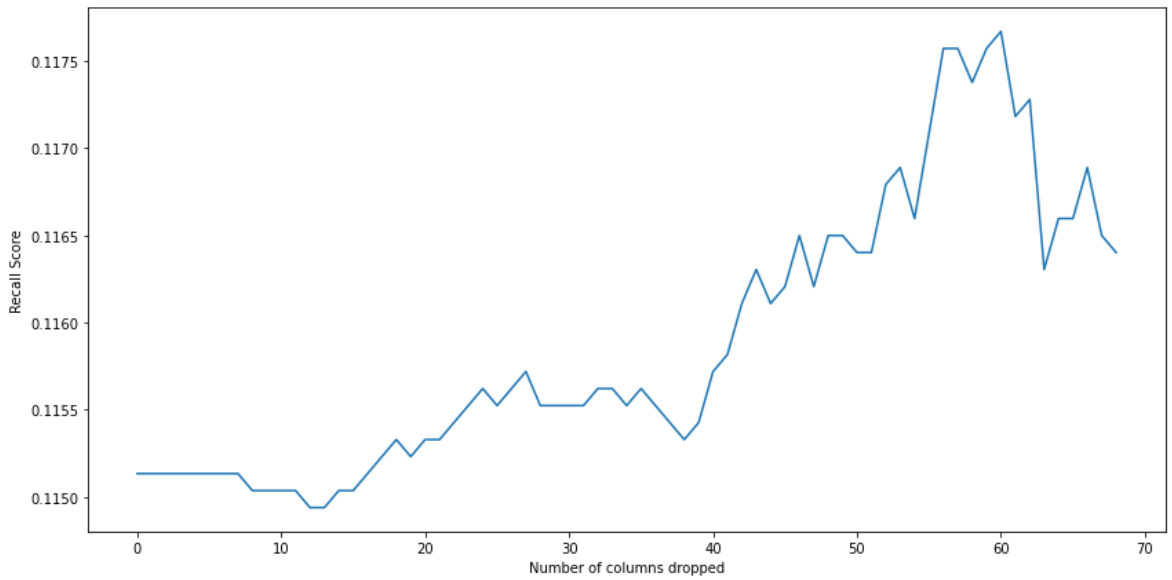
Inferences:

ROC-AUC Score



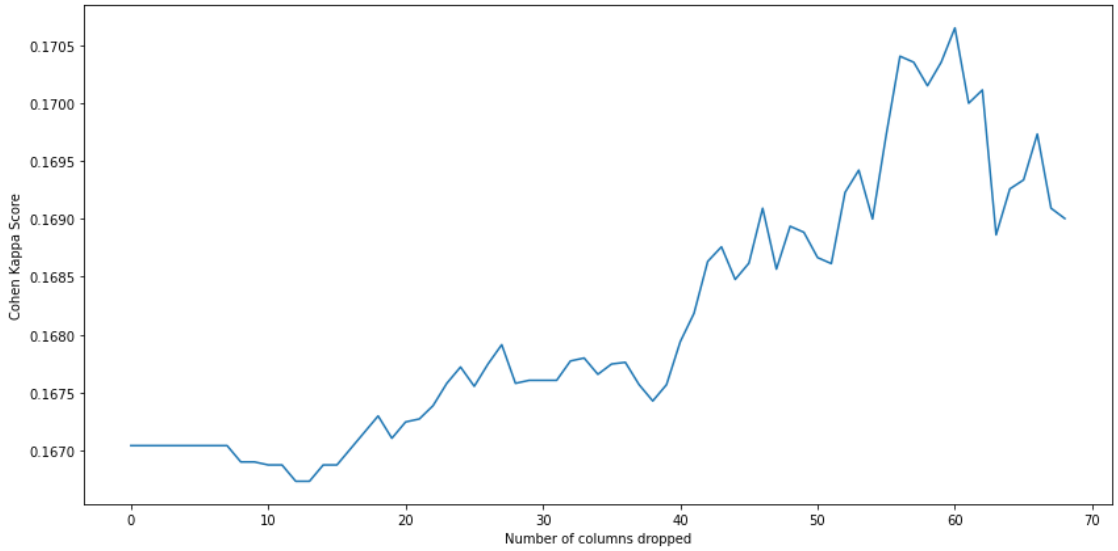
With elimination of each column/feature the ROC-AUC score gradually dropped lower

Recall Score



Whereas the Recall score , which is our primary validation parameter in our model building process, was increasing with the elimination of each feature.

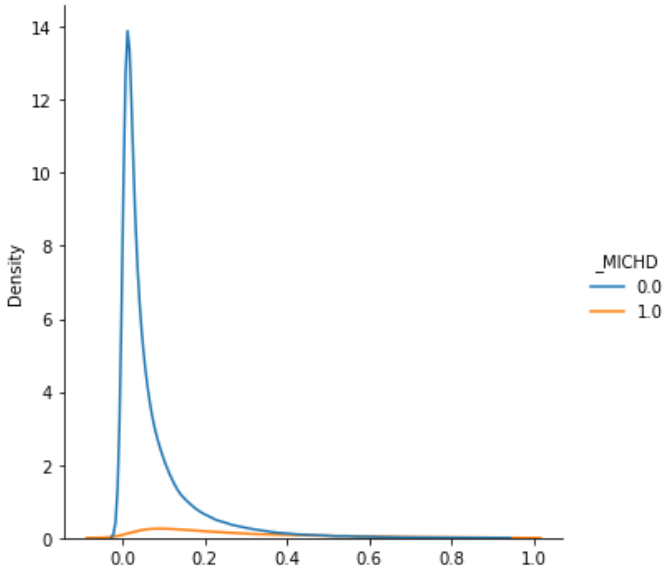
Cohen-Kappa Score:



The Cohen- Kappa Score also showed huge improvement with the elimination of the features.

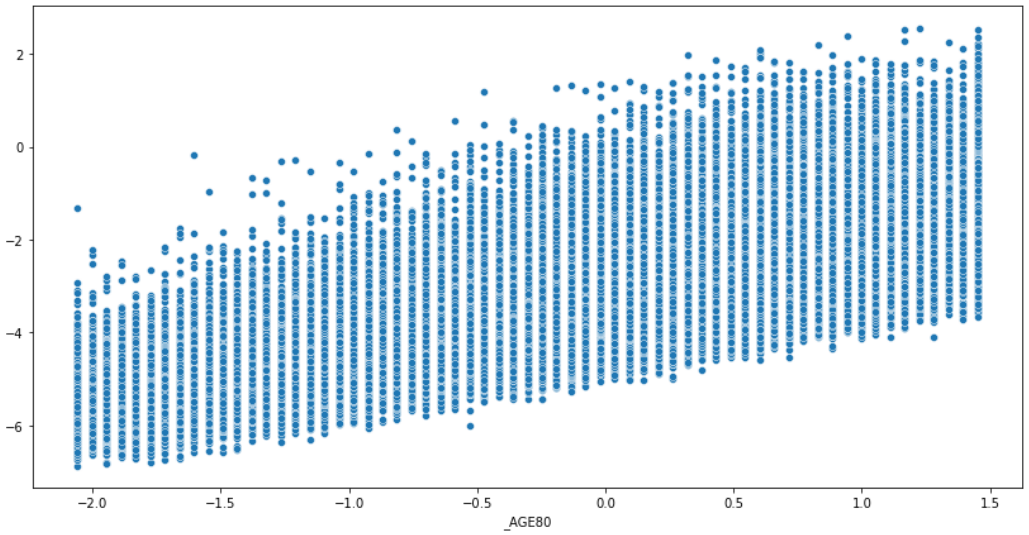
After which running logistic regression was implemented to check the results of the test dataset which actually contributed to better recall scores 0.35/1.0. Yet the classes were overlapping.

Overlapping of classes:



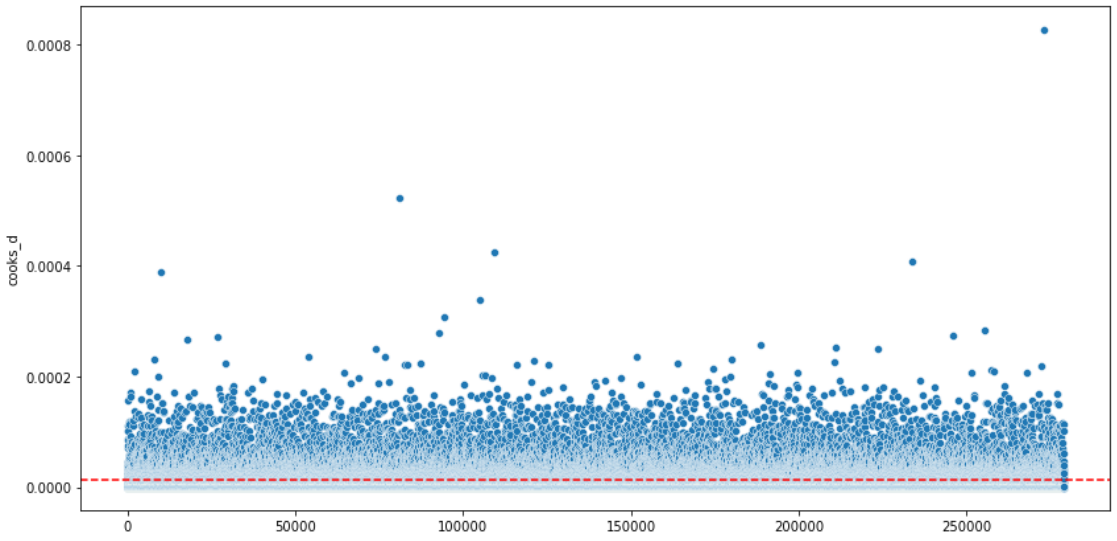
The Assumptions of the Logistic Regression had to be re-checked

Linearity was checked for the features, here the following is the linear relationship of the independent variable Age and the dependent variable MICHD, Yet few features like HTIN show no linearity.



Re-Check on the Outliers:

Here the task was to check for the extreme influential outliers and remove the for which Cooks distance approach was followed using GLM from stats model.



So the presence of extreme outliers could be visibly seen and it had to be properly treated so that the model could generalize better.

Hence a separate model was deployed eliminating the outliers determined by cooks distance.

Logistic Regression having No Outliers (Determined by Cook's Distance)

The results were ,

Classification report on Train set of no outliers

	precision	recall	f1-score	support
0.0	0.95	0.99	0.97	254952
1.0	0.51	0.21	0.29	17294

Classification report on Test set of no outliers

	precision	recall	f1-score	support
0.0	0.93	0.99	0.96	109268
1.0	0.53	0.15	0.23	10249

Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Logistic Regression, removed Cook's outliers	0.149966	0.915686	0.203606	0.79375	0.007112	0.842969	0.532088	0.030128

Inference:

The model seems to be slightly overfitting and the variation in train and test scores is lesser at the cost of the overall model performance and also the bias and variance errors have increased.

Grid Search-Tuned Logistic Regression

Classification report

	precision	recall	f1-score	support
0.0	0.93	0.99	0.96	109268
1.0	0.53	0.15	0.23	10249

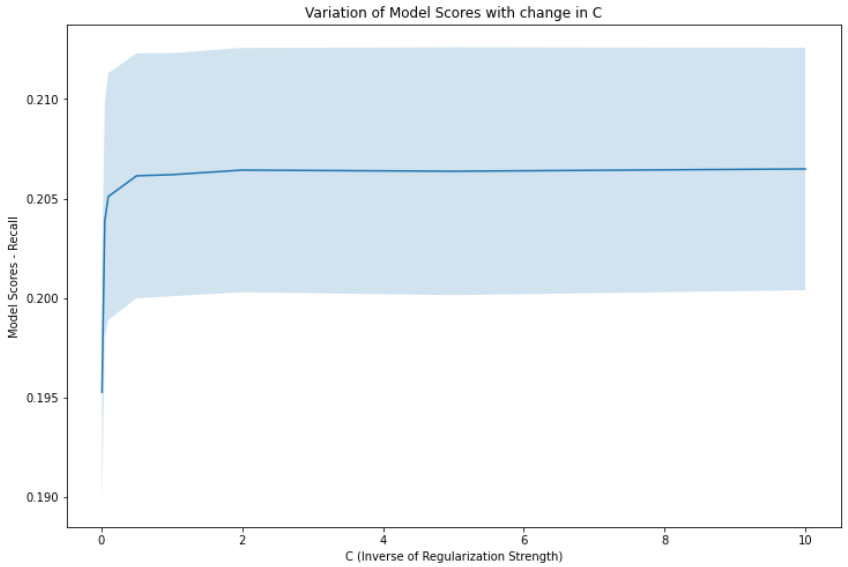
Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
Logistic Regression, C=10	0.150063	0.915694	0.203737	0.79375	0.007112	0.842969	0.532107	0.030129

Inference:

By running a Grid Search and changing C=10 (inverse of regularization strength), we see that Recall score and Cohen Kappa Score have slightly increased.

Visualization of Model Scores with Change in C



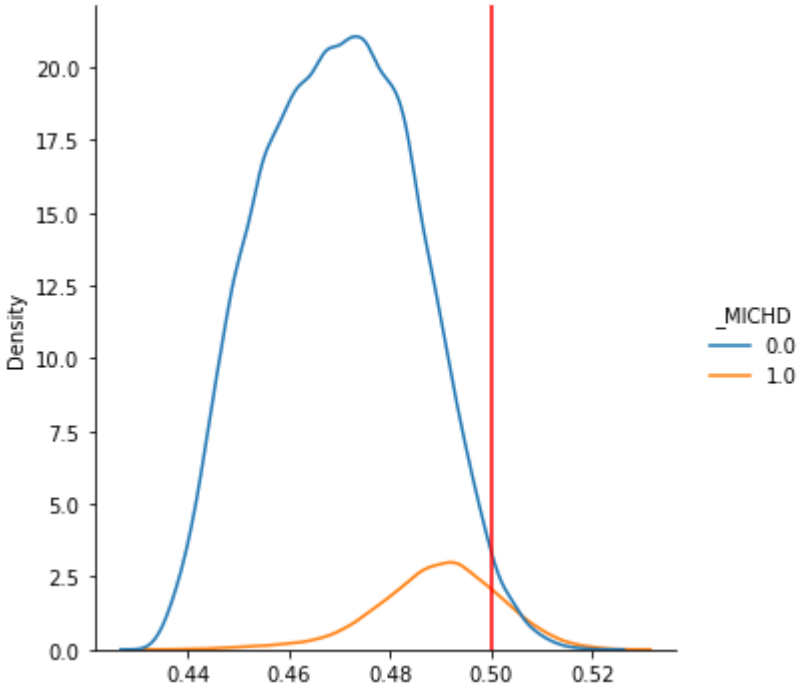
Inference:

According to the above graph, increase C makes the model's score increase, but also increase the model's variance error.

Ada Boost Logistic Regression

Classification report on Test set

	precision	recall	f1-score	support
0.0	0.93	0.98	0.95	109268
1.0	0.50	0.19	0.27	10249



score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=10] - default	0.189189	0.914029	0.237954	0.737328	0.005128	0.84186	0.529978	0.480354

Inference:

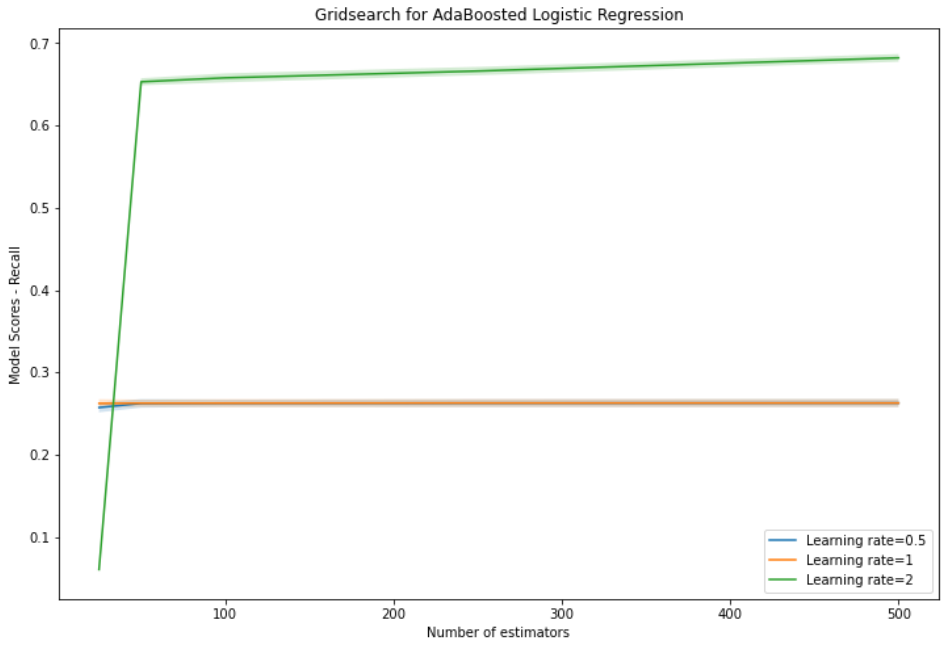
The AdaBoosted Logistic Regression with (C=10) has a much better recall score than just simple Logistic Regression

As we could see that the ada boost model to be the pretty good fit , we'll run ada boost with tuning and with various different parameters to arrive at best result possible.

Grid Search Tuned AdaBoosted Logistic Regression

The range of estimators given , params = {'n_estimators': [25, 50, 100, 250, 500], 'learning_rate': [0.5, 1, 2]}

The best parameter fetched by tuning is {'learning_rate': 2, 'n_estimators': 500}



From the above graph We see that the AdaBoosted model with 500 estimators is the best, with learning_rate=2

Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=5]: n_estimators=500, lr=2	0.495561	0.874043	0.335239	0.318084	0.005696	0.841885	0.530138	0.499476

Inference:

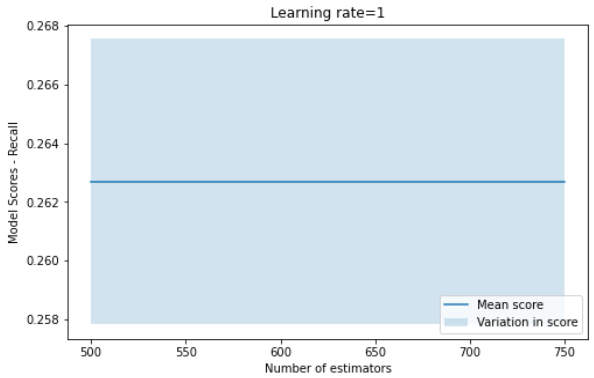
There is a significant improvement in the Recall and Cohen Kappa scores, while the ROC-AUC score has not improved much.

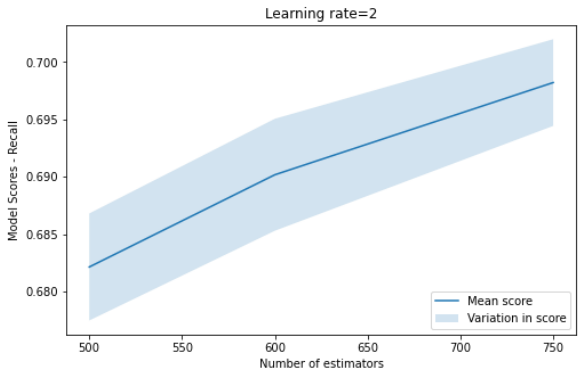
Grid Search Tuned AdaBoosted Logistic Regression 2

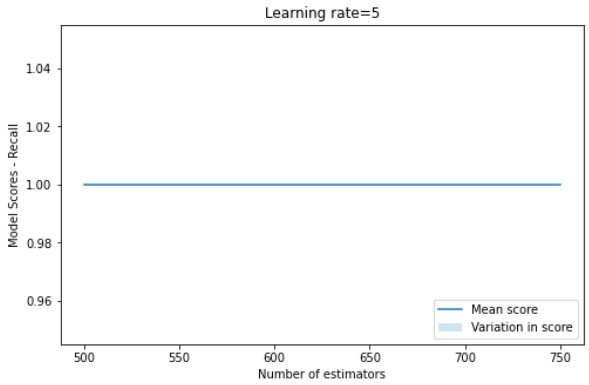
The range of estimators given , params = {'n_estimators': [500, 600, 750], 'learning_rate': [1, 2, 5, 10]}

The best parameter fetched by tuning is {'learning_rate': 5, 'n_estimators': 500}

But based on the visualization for each learning rate:







Inference:

- It's not recommended to pick the best suggested by the tuning

- We also see that with higher learning rates, the models get increasingly more unstable.

- Maybe the best combination to pick is with learning rate=2 and number of estimators around 600

Therefore maximizing the estimator range to 750 and the best estimator fetched is also 750 with learning rate 2

Classification report of the Test dataset:

	precision	recall	f1-score	support
0.0	0.98	0.90	0.94	254952
1.0	0.33	0.70	0.45	17294

Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=10]: n_estimators=750, lr=2	0.504927	0.871458	0.333805	0.302251	0.004947	0.841887	0.53012	0.49966

Inference:

There is a significant improvement in the Recall and Cohen Kappa scores

Grid Search Tuned AdaBoosted Logistic Regression 3

The range of estimators given , params = {'n_estimators': [50], 'learning_rate': 2}

Classification report on the Test Dataset

	precision	recall	f1-score	support
0.0	0.95	0.92	0.93	109268
1.0	0.35	0.47	0.40	10249

Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=5]: n_estimators=50, lr=2	0.470095	0.879557	0.335635	0.354732	0.006524	0.841822	0.530157	0.494283

Inference:

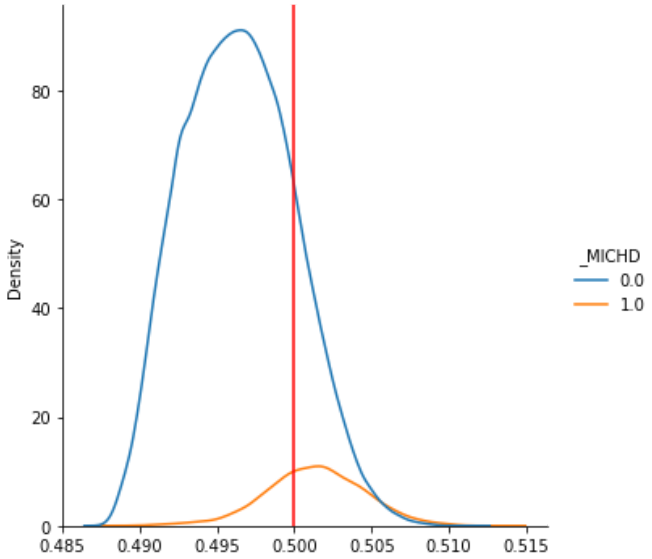
There is a slight decrease in the Recall and Cohen Kappa scores compared to the previous estimator , but building the next model with data that contain outliers would aid in a better model.

Grid Search Tuned AdaBoosted Logistic Regression 4

The range of estimators given , params = {'n_estimators': [50], 'learning_rate': 2}

Classification report of the Test Dataset

	precision	recall	f1-score	support
0.0	0.96	0.83	0.89	109268
1.0	0.27	0.67	0.39	10249



Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=5]: n_estimators=50, lr=2 - fitted with outliers	0.667577	0.817917	0.300809	0.330591	0.008633	0.844171	0.53523	0.498681

FINAL MODEL : Grid Search Tuned AdaBoosted Logistic Regression

The range of estimators given , params = {'n_estimators': [750], 'learning_rate': 2}

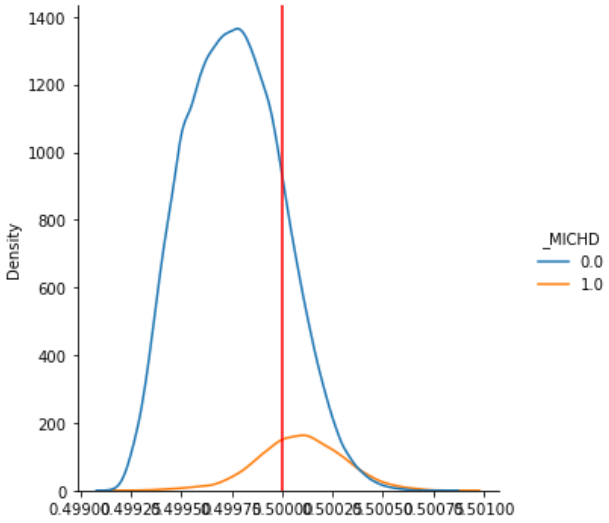
Code:

```
ada_lr_tuned_5 = AdaBoostClassifier(base_estimator=LogisticRegression(C=10),
                                    n_estimators=750,
                                    learning_rate=2)

ada_lr_tuned_5.fit(X_train_full, y_train)
```

```
AdaBoostClassifier(base_estimator=LogisticRegression(C=10), learning_rate=2,
                   n_estimators=750)
```

Classification report on the Test set :

	precision	recall	f1-score	support
0.0	0.96	0.83	0.89	109268
1.0	0.27	0.67	0.39	10249



Score:

	Recall Score, pos=1	Accuracy	Cohen Kappa Score	Bias Error	Variance Error	ROC-AUC score	Best Youden	Best Threshold
AdaBoosted Logistic Regression [C=10]: n_estimators=750, lr=2 - fitted with outliers	0.668553	0.817147	0.29994	0.328893	0.009021	0.844137	0.534426	0.4999

Inference:

- The best Logistic Regression models are the AdaBoost Models

- We see that there is very little difference between the model with n_estimators=50 and n_estimators=500.

- In favour of increasing model performance in terms of time taken, we will finalize the AdaBoosted Logistic Regression with n_estimators=750

- Further, we see that the models that include the outliers in the data perform better than the ones without the outliers.

CHAPTER 6 : Limitations ,Conclusion and Future Work

Limitations

Final Model: AdaBoosted Linear Regression

- As AdaBoosted Linear Regression is an ensemble model, the interpretability of the model is very complex. Thus, there is no clear relationship between the model and the features collected from the patients. Nevertheless, we do obtain a sense of the features important to the elevation of risk in the patients.
- Many portions of the data had to be dropped as the dimensionality was too high, along with the missing values in the data. Although this was optimized to the fullest, this could have led to a loss in information present in the original dataset.
- The results of the model are indicative of whether or not the patient is inclined to develop Coronary Heart Disease. This is not a clear indication of the risk to the patient's life.
- As the model's ROC-AUC score was not perfect, there will be patients who are not under the risk of developing heart disease who will be categorized as such. By lowering the threshold (using Youden's Index) to find the best possible recall, we prioritized finding the patients with risk of heart disease over falsely identifying those that did not have it.
- The dataset contained many complex relationships between the response variable and the predictors which led us to choose complex models over easily interpretable ones.

Scope

- The model probably cannot be used as a standalone application to predict the chance of developing coronary heart disease.
- This sort of model, once stable enough, could be deployed in clinics and research centers which would serve as future test and training data to better predictions. This type of model is best used along with a medical practitioner's own intuition to empower their decisions with the help of statistics.
- Further, the model only predicts whether an individual has a chance of developing heart disease. It does not answer the following questions: When will the patient show symptoms of coronary heart disease? Is the patient's life at risk at the current point in time? What factors contributed greatly to this development of risk for the patient?

Closing Reflections:

- The original dataset contained many types of columns, including sub datasets that we couldn't explore fully. Sub datasets included questions to pregnant women, diabetic patients, females with cancer, etc. This data could lead to new findings in the relationship between coronary heart disease and the subset of people.
- Deep Learning and Stacked Models are another possible path to try for training this data, although at the cost of interpretability.

Conclusions and Future Work:

- The dataset provides many opportunities and domains of exploration. For example, we could study finer patterns of the risk of heart disease within the subset of pregnant women, diabetic patients, etc.
- The model could be a multi-branched algorithm that predicts using different models depending on which subset the person belongs to.
- Once complete, this project could be deployed using a low-code / no-code application, like Streamlit, and exposed to real world test data.

BIBLIOGRAPHY

1. Machine Learning: A Probabilistic Perspective – by Kevin P.Murphy.
2. Introduction to Statistical Learning using R by Hastie and others
3. Course textbooks etc.
4. <https://www.kaggle.com/code/nkitgupta/advance-data-preprocessing>